FB9401V2

EOSDIS Core System
Science Information Architecture

White Paper
Working Paper

March 1994

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Bruce Moxon, SDPS Architecture Team — Date
EOSDIS Core System Project

SUBMITTED BY

Steve Fox, SDPS Office Manager — Date
EOSDIS Core System Project

Hughes Applied Information Systems, Inc.
Landover, Maryland

This page intentionally left blank.

Contents

6. ECS Sub-Architectures

7. Risk

8. Future Directions

9. References

Figures

Tables

Abbreviations and Acronyms

1. Introduction

1.1 Purpose

This document presents a Science Information Architecture for the Earth Observing System Data and Information System (EOSDIS) Core System (ECS). It is based on key components of the conceptual architecture presented at the EOSDIS Progress Review, held December 13-14, 1993 in Landover, Maryland.

The Science Information Architecture focuses on those portions of the system involved in the production, management, and distribution of science data products. Hence, it reflects largely a Science Data Processing Segment (SDPS) focused view of the overall EOSDIS system. It calls upon, and interacts with, portions of the Communications and System Management Segment (CSMS) and the Flight Operations Segment (FOS). Those segments are referenced herein to the extent that they interact with Science Information architectural components.

The white paper presents the following key concepts in developing the Science Information Architecture:

- an overall system context, based on a number of science-based architecture drivers
- a set of fundamental design principles
- a high-level conceptual architecture based on these drivers and assertions
- a first level logical decomposition of functionality into a set of complementary sub-architectures and their composite service classes

This is the second revision of the Science Information Architecture white paper. The first revision was entitled “ECS Science Data Processing Sub-architecture”. The title was changed to reflect this document’s presentation of a broader scope than the term “science data processing” implies. We attempt to address here all aspects of producing, managing, and distributing science information – which may include data, metadata, code, reference materials, and more.

This paper develops the architectural framework, and presents a first level decomposition of the system into a set of complementary sub-architectures and associated service classes. The paper serves to set an architectural “direction”, with additional detail to be developed within that framework as system design efforts continue. It should be noted that the development of the architecture is proceeding in parallel with system design activities. The architecture presented in this document, therefore, represents work in progress and is subject to change.

Finally, this document provides the background and architecture context for the program in support of the baseline system architecture review. It will serve as contextual material for the System Design Specification, DID 207, which will be issued in conjunction with the System Design Review (SDR) in mid-1994.

Acknowledgments

The material for this paper was developed through the efforts of a number of ECS Project Staff members. Particularly valuable were the contributions and comments from members of the SDPS Architecture team, including Mark Elkington, the team leader, Mike Burnett, Eric Dodge, Jolyon Martin, Richard Meyer, Carl Wheatley (representing CSMS), and Ron Williamson. Steve Fox and Erich Stocker were instrumental in providing direction and guidance, and in arranging for additional team support.

1.2 Organization

This paper is organized as follows:

- Section 1 presents the purpose and scope of the document, its organization, and logistics concerning its review.
- Section 2 establishes the context of the Science Information Architecture, including its background and origins and a list of relevant resources.
- Section 3 presents key system drivers obtained from ongoing discussions with the science community.
- Section 4 extracts an architectural mandate from the system drivers, and develops an architectural philosophy and approach that brings together key science-based drivers and fundamental system engineering principles.
- Section 5 presents a conceptual architecture that embodies the mandate and approach presented in section 4. The conceptual architecture develops a layered model of high level system functionality, and illustrates the relationship between that functionality and the operational institutions within EOSDIS.
- Section 6 presents a set of logical sub-architectures, decompositions of the conceptual architecture that focus on key aspects of the system.
- Section 7 explores some key architectural challenges and risks, presented in the three-layered structure of the conceptual architecture.
- Section 8 concludes the discussion by setting this document into an overall system analysis and design context, and discussing future steps.
- Section 9 presents a set of references.

1.3 Review and Approval

This White Paper is an informal document approved at the Office Manager level. It does not require formal Government review or approval; however, it is submitted with the intent that review and comments will be forthcoming.

Questions regarding technical information contained within this Paper should be addressed to the following ECS and/or GSFC contacts:

- ECS Contact

  Steve Fox, SDPS Manager
  (301) 925-0346
  sfox@eos.hitc.com

- GSFC Contact

  Erich Stocker, SDPS Technical Manager
  (301) 286-2153
  estocker@gsfcmail.nasa.gov

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Applied Information Systems, Inc.
1616A McCormick Dr.
Landover, MD 20785

2. Background

2.1 Architecture Definition Process

The Science Information Architecture white paper reflects a significant change in the architectural direction of ECS, brought about by the activities surrounding the ECS System Requirements Review (SRR), held September 13-14, 1993 in Greenbelt, Maryland. The status of these activities were reported during the ECS Progress Review in December. This paper is aimed at documenting the new architectural direction, and providing a framework for continuing refinement and system design.

In addition to the architecture material presented at the ECS Progress Review in December, the architecture incorporates drivers from a number of parallel activities, including:

- the GCDIS/UserDIS study
- SRR Feedback and recommendations of the Data Panel
- development of Science-based Architecture Drivers through ongoing visits by members of the ECS Science Office to a variety of investigators' sites
- continuing analysis of the Version 0 capabilities and architecture
- an Evolution and Evolvability Study
- continuing development of the User and Data Models
- initiation of a number of External Systems Studies to identify and incorporate, as appropriate, approaches and architectural concepts from a number of current information systems and applications, including:
  - Sequoia 2000
  - NIIT Earth Data System (EarthDS)
  - WAIS
  - World Wide Web and Mosaic
  - a variety of corporate information systems
- continuing efforts in project-directed architecture analyses and trades

After the ECS Progress Review in December, the architecture effort was expanded to develop the conceptual architecture into a set of complementary logical architectures. This process, depicted in Figure 1, is ongoing.

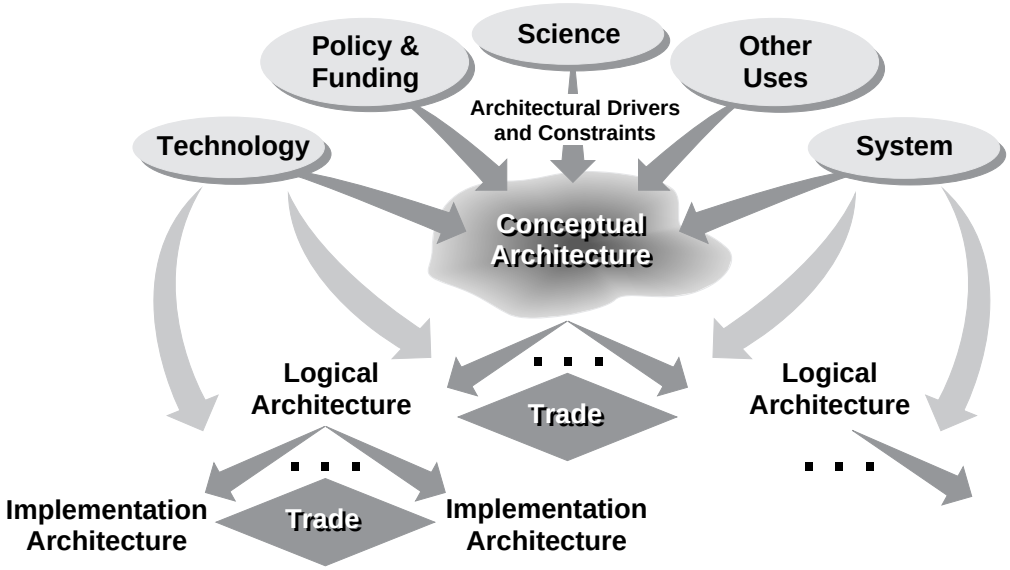

Figure 1. Architecture Definition Process

We define a logical architecture[1] to be a structured decomposition of a portion of the overall architecture, with focus on the components and services in that decomposition. An implementation architecture then presents the mapping of the logical components to a physical system configuration (hardware and software components), providing the realization of the physically distributed system. In this paper, we focus on the development of the conceptual architecture, and an initial decomposition into a set of logical sub-architectures.

Many of the investigations into the architecture drivers are also continuing in parallel with the architecture definition activities. These drivers and constraints (resources, budgets, and appropriately scoped objectives) guide the required policy and technology decisions that result in a specific implementation of the "policy neutral" conceptual architecture. As additional drivers and constraints are identified, they will be used to guide the development and evaluation of the logical and implementation alternatives.

The results of this continuing analysis and design will be reflected in the logical and implementation architectures that will be presented at the System Design Review in mid-1994.

1 We use this term interchangeably with "logical reference architecture", or more simply "reference architecture".

2.2 Related documents

This document focuses on the rationale for the Science Information Architecture, and the architecture itself, with minimal background discussion. It assumes familiarity with the concepts developed in related project documents, including:

93-216-SE1-001	EOSDIS Core System (ECS) Requirements Specification , August 1993 (in revision)
193-202-SE1-001	Standards and Procedures for the ECS Project, August 1993
193-604-OP1-001	ECS Operations Concept Document for the ECS Project, August 1993 (in revision)
193-00611	Science-based System Architecture Drivers for the ECS Project White Paper, December 1993
193-00623	ECS Evolutionary Development White Paper, December 1993
193-000626	GCDIS / UserDIS Study White Paper, January 1994
FB9402V1	ECS Science Requirements Summary White Paper, February 1994
194-206-SE2-001	Version 0 Analysis Report, Working Draft, February 1994

Many of the concepts developed in the Science Information Architecture draw upon concepts in the "GCDIS / UserDIS Study" white paper. Rather than replicating the discussions from that study in this paper, we attempt to summarize key points and provide references to the appropriate sections of the white paper for further background.

3. Architecture Drivers

In this section, we present a review of the key drivers and influences on the system architecture. These drivers have been developed through interactions with the science community, and represent factors which may determine and affect science users' information needs. For purposes of discussion, we decompose these drivers into the following four categories:

- Science Drivers
- Technology Drivers
- System Engineering Drivers
- Policy and Funding Drivers

The following subsections provide a high level characterization of the driver categories, and highlight key points as they affect the architectural discussion that follows. The drivers themselves are addressed in greater detail in a number of the references enumerated in section 2.2.

We expect additional drivers to be uncovered in the ongoing user and data modeling activities, and in continuing interactions with the scientific community, including the Version 0 developers and users. We expect these drivers to further enlighten our understanding in a number of areas, enhancing our current thinking, but remaining consistent with our overall architectural direction. To this end, the Science Information Architecture described here provides a mechanism for evaluating and assimilating these new drivers into the system architecture.

3.1. Science Drivers

The science drivers are dictated largely by the research scientists' needs in supporting long term Global Change research. They are centered around the needs of research teams to effectively and efficiently find, access, use, and share the results of their research, which include data, modeling results, and observations and analyses. The science drivers are derived from various inputs from the scientific community, including feedback from SRR and visits with research teams associated with Science Computing Facilities (SCFs). They include:

1. Facilitate an efficient data search and "access" paradigm
2. Support a dynamic product life cycle and easily extensible product set
3. Support an interactive investigation capability
4. Support an information-rich data pyramid
5. Support the integration of independent investigator tools
6. Support user-to-user collaboration
7. Provide distributed administration and control to support site autonomy

These science drivers are explored in more detail in the “Science-based System Architecture Drivers for the ECS Project” white paper.

3.2. Technology Drivers

Technology drivers are based on advances in computer and communications technology that impact users’ abilities to search, access, and process information in a large system such as EOSDIS. These drivers focus not on the technologies themselves, but rather the ways in which evolving technologies affect the way users of the system will work. These drivers affect all potential users of the system, but especially researchers, as they will likely be most demanding in their day-to-day activities on the system. These drivers will have a wide ranging effect on how the system will promote or limit future global change research. We believe the following to be the more significant technology drivers affecting ECS:

1. Advances in the fundamental software infrastructure of an evolving information technology industry. These include developments in:
    - Operating systems that support network-based distributed computing
    - Distributed processing and interoperability protocols
    - Object-oriented Databases and object-extended relational models
2. Networking advances, including the potential for ubiquitous Gigabit communications based on advancing network technologies like ATM, SONET, etc.
3. Processing advances, including the continuing performance increases anticipated in desktop workstations, the clustering of such workstations in cooperative problem solving, and the next generation of MPPs.
4. Advances in multimedia technology which promises desktop videoconferencing and sophisticated workstation-based collaboration environments.

Key technology advances are tracked and analyzed in the activities of the Technical Assessment team, which reports regularly on its findings.

3.3. System Engineering Drivers

System engineering drivers dictate the system development effort’s response to issues like system scalability and evolvability, and to system reliability, maintainability, and availability. The drivers determine the extent to which the system must evolve over time, and hence provide design guidelines that will help developers determine an appropriate level of effort to devote to system expansion and flexibility.

A number of these drivers are explored in the “ECS Evolutionary Development White Paper”.

3.4. Policy and Funding Drivers

Policy and funding drivers are based on fundamental mission objectives and funding constraints, and include guidelines regarding expansion of the ECS system to support other uses, including GCDIS/UserDIS , support of value-added providers, and commercial use of the system.

The intent is to develop an architecture that can adapt to a range of policy and funding drivers that may change over time, i.e., a "policy neutral" architecture. While we typically think of policy and funding issues as long time frame drivers, there will most certainly be a need to adapt to shorter time frame strategic missions as well. As demands for various data products change over time, the system must be flexible enough to accommodate those changes. This may include redirecting some portion of product generation computing resources, or employing additional resources from, say, an NSF supercomputing center, to the production of key research products. Such activities might be undertaken in response to a phenomenological event.

The "GCDIS / UserDIS Study" white paper presents one dimension of this driver category, in exploring the needs of a larger community than ECS directly supports. The study explores the concept of an all-encompassing Earth Science information *federation*, based on complete subsystems like ECS, and linked into a larger information "highway". The study attempts to establish a larger context of which ECS is a part, in order that ECS might be able, through its development, to establish standards and components that could be adopted by the global change community at large. This is accomplished through careful analysis of both the significant similarities and differences between the uses of ECS and GCDIS / UserDIS.

There are many similarities between the overall objectives of GCDIS / UserDIS and ECS, and in fact the ECS Data Management Architecture described in this paper draws a great deal from the GCDIS / UserDIS intersite architecture. However, there are also some significant differences between the objectives of ECS and those proposed for GCDIS / UserDIS, many stemming from the fact that ECS is a funded project, with a defined authority, responsibilities and specific set of organizations involved in its definition and development. This enables, and in fact requires, ECS to exert greater influence and control over the system direction than might be afforded the GCDIS community.

Some of the key differences brought about by this different landscape include:

- the need to define a site architecture, in addition to an intersite architecture

  ECS includes a number of data "authorities", through the instrument teams and the Distributed Active Archive Centers (DAACs). These groups, through appropriate community oversight, will provide data to the ECS community in program prescribed ways. This allows for, and in fact requires, the definition of conforming site architectures that act as both data providers and consumers within ECS.

- the need to adopt and develop key standards

  Whereas GCDIS must leave open areas involving data standards, because of its lack of a governing body, ECS can and will identify and drive standards which it believes are key to the fundamental success of its mission. These standards might include specific data formats, protocols, and languages that are used within ECS as a foundation of system

operation. Such standards are not only desirable within ECS, but necessary to ensure compatibility across instrument datasets, and to reduce redundant effort in tool development.

However, to meet its goals of evolvability, ECS will also need to support interoperation with systems that may not have adopted the same standards. In that sense, many of the objectives of the GCDIS / UserDIS intersite architecture are in line with those of the ECS architecture. While ECS will allow and support such interoperability, e.g., through the publishing of standards and protocols, and availability of reusable system components, it is will not implement a fully interoperable GCDIS.

- the need to balance user's retrieval needs with the system's archival responsibility

  ECS must be developed to balance user's information retrieval needs with fundamental data acquisition and archival needs, as it is defined as the repository for EOS data. GCDIS / UserDIS, on the other hand, currently has no specific data acquisition charter and hence is almost entirely focused on data access and analysis.

These differences highlight the need to focus ECS efforts on defined mission objectives, being mindful of and, where possible, accommodating wider needs. Because of the similarities in information support and the overlapping user communities, the extent to which ECS is successful in developing flexible, reusable components at both the site and intersite levels will determine its success in directly supporting extension to a GCDIS / UserDIS environment.

ECS-specific policy and funding drivers are discussed in more detail in the "ECS Science Requirements Summary"; GCDIS/UserDIS and its drivers are explored in the "GCDIS /UserDIS Study" white paper.

4. Architectural Mandate and Philosophy

In this section, we develop an architectural "mandate" based on the drivers presented in Section 3. We then develop an architectural philosophy and approach that provide the framework for a conceptual architecture that responds to this mandate.

4.1 Architectural Mandate

In response to science community and NASA direction, and as a result of post-SRR discussions and analyses, the ECS project has developed what could be interpreted as a "mandate" from the scientific community to develop a new conceptual architecture capable of meeting the needs embodied in the aforementioned drivers. Together, the elements of the mandate comprise the building blocks for an evolvable system.

That mandate is summarized in the following set of system design guidelines.

	Move from ...		*Towards ...*
Ø	Product approval and ordering	√	Product "publishing" and "access"
Ø	Metadata / data distinction	√	Seamless view of all data
Ø	Limited provider implementation	√	Extended provider implementation
Ø	Homogenous, centrally managed system components	√	Heterogeneous, autonomous system components

This last point incorporates a number of related issues, including the move from an element-partitioned architecture to a service-oriented architecture, support for a distributed system composed of heterogeneous components, and a decentralized (both physically, and in terms of authority) system management approach. An expanded version of this mandate covering these and other points is explored in detail in the "Science-based System Architecture Drivers for the ECS Project" white paper.

4.2 Architectural Philosophy

In the previous section, we presented a science-derived mandate that included a set of architectural imperatives. In this section, we present a system engineering-based architectural philosophy that embodies these imperatives, as well as additional guidelines that we have selected to govern the design and development of the SDPS architecture. These guidelines are based on conventional system engineering wisdom, and the development of approaches based on emerging information technologies.

In the following subsections, we explore some of these key principles as background for the development of the conceptual architecture, which follows.

4.2.1 An Open Distributed Architecture

There are a number of efforts underway to develop the foundations for "open distributed architectures". Such architectures are characterized by the ability to easily integrate physically distributed heterogeneous components into a single seamless system. The ISO/IEC standard-in-progress for Open Distributed Processing, or ODP (Pochardt-Johnson, 1992), for example states that:

> *"Openness allows for the integration of heterogeneous components. This may involve equipment from different vendors, running different operating systems, with applications written in different programming languages, running different database engines, and with different security authority levels".*

This philosophy captures the intent in developing an open distributed ECS Science Information Architecture. An open distributed architecture, such as that defined by the ODP standard:

- enhances application portability, by building on a layer of common, well-defined interoperability interfaces.
- promotes proper operation in a heterogeneous computing environment.
- enables incremental system evolution by abstracting system functionality in service interfaces, and by encapsulating the underlying implementations.

The ECS Science Information Architecture is a variant of existing and emerging industry approaches (often embodied in "Reference Models") governing the description of open distributed computing systems. Its purpose is to provide a framework for system decomposition that is used to guide analysis and design activities. The decomposition highlights the interfaces between distributed computing components, and allows us to focus on existing and emerging standards for interoperability. This insures that EOSDIS will evolve with advances in distributed computing technology to provide a state-of-the-art information system to its users.

4.2.1.1 The Client-Server Model

The Science Information Architecture is based on a fundamental client-server model. This model is illustrated in Figure 2. In this model, a *service* is a collection of related functions which are accessible via program callable interfaces across a network. A *service class* is the generic definition of these functions and their interfaces. A *server* is a process[2] which instantiates a service class.

A *client* is a process requesting the execution of one of the service functions. The request is called a *service request*, and the interaction between the client and the server is called a *client-server interaction*. A server can itself call other servers while processing a client request. Such a server is said to be acting as an *agent* of the client.

---

2 Here, we use the term "process" loosely. At the level of abstraction we're concerned with, a server might be implemented by one or more operating system "processes", perhaps running across multiple machines, such as in the case of a distributed database server.

The client submits a service request via a *client interface*. The client-server interaction establishes a connection between this interface and the corresponding server interface to transport the service request and return the service request result.

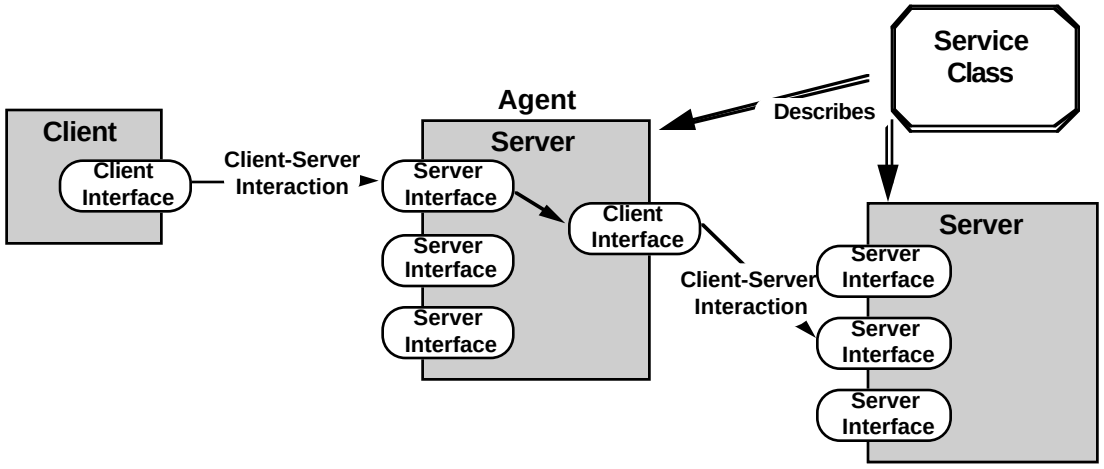

Figure 2. Basic Client Server Model

In a large system like ECS, the network will need to support many instances of a service class, both for performance reasons, and because of distributed operations drivers. In such a situation, a client may request to be connected with a specific server, or it may describe its requirements on a more generic level. This raises the issue of finding the server which the client requested, or which satisfies the requirements specified by the client.

4.2.1.2 Services and Objects

The issues associated with distributing and locating instances of a service class has spawned significant developments in technologies for distributed information systems. These technologies, in general, provide a buffering layer between clients and servers as defined in the traditional client-server model. This buffering layer provides additional flexibility in constructing distributed systems, by providing dynamic routing and handling of a client's service request.

There are a number of promising technology advances in this area, many of which are based on object-oriented technologies.[3] However, the speed with which these technologies will mature is not known. In order to facilitate the transition from traditional client-server distributed computing implementations, to object-oriented implementations, the Science Information Architecture will be captured in an objects- and services- based description.

---

3 For example, the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) (OMG, 1991)

Information systems are often described in terms of the information "services"[4] they provide to their users. This is a natural way to describe information transactions between users and systems. The service metaphor is often extended to also describe the relationships between internal components within a system, again because of the focus on the transaction itself, rather than representation specifics of the underlying information.

Once we describe a set of services, we can discuss how system components request and provide those services. An object-based system description allows us to develop a model of both concrete and abstract system "objects" that may aid in how we conceptualize and organize system components.

When we talk about an "object-based" approach, we don't necessarily mean to imply formal object oriented methodologies or object oriented programming languages, although these do provide a good method for organizing and describing a complex system such as ECS. Rather, we refer to the notions of data *abstraction* and *encapsulation* that provide a framework for describing system components (both physical and abstract) in a coherent and uniform manner.

Encapsulation promotes a discipline in which the information associated with an object is only accessible through the services supported by that object. Hence it limits the access to an object's data to a small set of specific, well known interfaces. Abstraction separates the underlying representation of an object from its behavior (as witnessed through its services), allowing the representation to change without affecting the object's services.

A third property, *inheritance* allows for a hierarchical formulation of services, in which higher level services (object-specific service classes, or *specializations*) are developed as extensions to lower level services (*generic* service classes). This organization promotes the reuse of fundamental building blocks in developing large portions of the system.

In contrast with a purely functional decomposition, where functions are assigned early to subsystems and hence physical components, an object-based organization allows for greater latitude in combining and distributing portions of the system, leading to less reliance on specific physical subsystem components, and greater flexibility in system implementation. This flexibility allows for organization of system components (objects) in a manner that supports greater reuse of basic system building blocks in supporting higher level services.

4.2.1.3 Evolutionary Approach to Service Development

The encapsulation and abstraction concepts outlined above support an evolutionary approach to service, and hence system, development. Because system components are insulated from the implementation details of other components through their service interfaces, we are free to evolve the capabilities of components in an incremental fashion. The extent of changes to the system to accommodate things like more efficient data representations can be restricted to the definition of the objects themselves, thus limiting the scope of changes required to accommodate system evolution.

---

4 When we talk of "services" in this context, we sometimes refer to an abstract "service" provided to clients, as opposed to a specific service interface. In some sense, this is a behavioral description of a portion of the system, and may actually be composed of a number of actual service calls.

This allows for the development of increasingly sophisticated system components, and encourages third-party development of new services and system components. This evolution can occur both at the level of client application services (i.e., those seen directly by the user), and within infrastructure services (e.g., "finding" appropriate services).

For example, in early releases of the system, fundamental data "inventory" services based on static bindings to inventory service providers can be developed. In satisfying a user's request for inventory information, the system would look up a number of known inventory providers, and obtain the appropriate information from one or more of them. Later, when distributed object technologies that employ dynamic advertising and request brokering services are available, the system can evolve to take advantage of dynamic binding capabilities. This would allow new inventory providers to make their inventory information available without the need for complex, coordinated update of existing system data structures.

Applying the concept of inheritance to employ basic building blocks in a wide array of higher level services provides a kind of evolutional leverage, in that improvements in fundamental services can be propagated across a wide range of higher level services. For example, if we develop a number of higher level user services (data access, subscription, resource discovery) on fundamental distributed search services, and we evolve those distributed search services to provide more focused search capabilities, then it should be possible to propagate those improvements to the higher level user services.

There are undoubtedly other advantages of moving towards object oriented technologies. For example, the data types supported by conventional databases are not particularly good for supporting the wide array of data types required to efficiently represent earth science data. Here, a migration towards object oriented technology may greatly simplify some of the underlying data provision services. Therefore, our goal is to permit eventual migration to object oriented technologies as they mature. We are employing this evolutionary approach in three key areas to enable this migration:

- the development of an objects and services-based system decomposition, as described in section 4.2.1.2. In this approach, we define a logical object whose interfaces implement a collection of related services we call a service class.
- abstract interfaces to distributed computing components will support migration from a client-server based approach (i.e., employing remote procedure calls) to a distributed object approach (e.g., CORBA) with minimal impact on software.
- the database management and search architecture will enable a transition to an object paradigm, again with minimal impact on applications and users.

This and other examples of evolutionary service development are explored in more detail throughout the sub-architecture decomposition in section 6.

4.2.1.4 Standards and Technology Integration

One of the keys to leveraging the evolutionary framework described above is through the use of standards in supporting technology integration. However, the issue of standards is a complex one in any state-of-the-art information system. On one hand, network communities rely

extensively on the adoption of common conventions and standards for their successful interoperation. However, there seems to also exist a continual flow of new organizations developing new and often competing standards -- there is no single, universally accepted set of standards. For example, Internet conventions are often at odds with standards used in other communities, including ongoing international standardization efforts such as the OSI.

Commercial vendors often promote their own “standards”, either because the standards process lags behind their product development schedules, or because they can add value (performance or functionality) to a product by circumventing, modifying, or interpreting portions of a standard. As a result, products based on the same standard are often not truly compatible.

As a system, ECS will depend upon making optimal use of standards-based commercially available off-the-shelf software and hardware, and where appropriate, public domain software, or freeware. Hence, this standards landscape requires ECS to closely track and participate in the development of key standards where they exist, and to help push the development of new standards where none currently exist. Hence ECS is active in tracking key technology standards in areas such as distributed computing (OSF/DCE, CORBA), high performance computing (HPF and FORTRAN90), database technologies (SQL3), distributed file systems (OSF/DFS), and mass storage systems (IEEE Mass Storage Reference Model)[5]. Additionally, we will be helping to define and develop new standards in, for example, the definition of an earth science data language and earth science data formats. In such efforts, we will attempt to identify and team with organizations that have begun, or are doing, similar work in these areas. Put simply, the ECS mission in the area of standards is to:

- *adopt* appropriate existing standards
- *track* proximal proposed standards
- *contribute* to the development of promising standards efforts
- *drive* efforts to develop new standards where none exist

While ECS will have to rely on the adoption of community-wide conventions and standards for its ultimate success, the architecture also needs to isolate itself from the issues introduced by over reliance on standards or commercial products. This is done through application level abstractions on underlying standards-based services.

4.2.2 Unifying Concepts

In this section, we develop a core set of unifying concepts for ECS by overlaying the application context on the elements of the architectural philosophy put forth in the preceding discussion. These concepts are governed by a couple of key system development principles:

- Use a small set of good concepts
- Apply these core concepts as universally as possible

5 In fact, ECS is tracking a large number of standards through its Technology Assessment program (ECS 193-202-SE1-001, 1993)

The application of these principles to system development reduces system complexity by decreasing the number of protocols and interfaces in the system, ensures interoperability between system components, and supports component reuse.

The three key concepts, as they relate to the development of ECS, are explored briefly in the following subsections.

4.2.2.1. Data = Data

This concept attempts, among other things, to remove the arbitrary distinction present in many systems between different levels of metadata, and between metadata and data. Historically, the distinction between data and metadata has been determined by a hard-and-fast rule: metadata is small, and searchable, hence stored in a relational database; data is large, and generally treated as a single object, not necessarily further decomposed in a search. However, this can lead to significant problems in a system as large and long-lived as ECS, including:

- one person's data is another person's metadata

  For example, masks can be viewed simply as image overlays, used in compositing for display purposes. Alternatively, they might be used in spatial relation operators (i.e., "where") to subselect data based on element-wise mask values. In this latter case, the data values within the mask "image" are actually used in element-wise relational operations to subset or subselect another image.

- metadata changes over time

  Over time, data may be re-characterized in a number of different ways (i.e., the metadata descriptions may change), for example as more is learned about a particular sensor's data. The ability to redefine what conceptually amounts to metadata over time is important for long term viability of datasets.

By treating metadata and data as logically equivalent entities, we simplify the data access model (there's a consistent way to get to all data), and provide implementation-transparent data access to users. This frees us up to represent data as needed to support anticipated operations, without making that representation apparent to the user.

This approach eliminates the impact in traditional systems of data "moving across the boundary". Such a reorganization can often have far reaching effects, in the reformatting of databases, and the modification of tools that provide access to the data. The concept is illustrated in Figure 3.

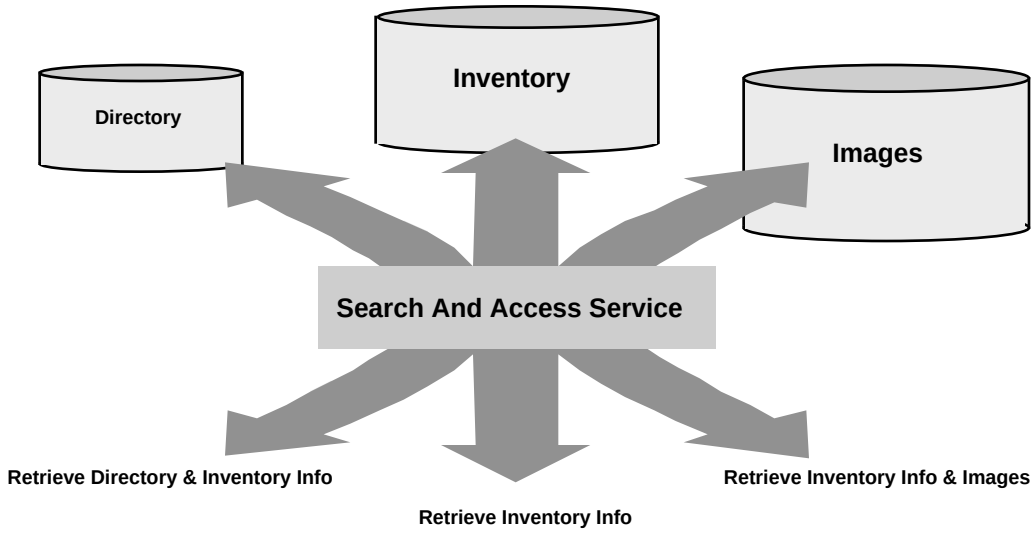

Figure 3. Data = Data

4.2.2.2. Data = Services

A similar concept is applied to blur the distinction between data and services. This is done by making data available only indirectly via services that provide it (the property of encapsulation). This is important because the optimal manner for providing data to users may be dataset and product level specific, and may change over time. For example, it may be cheaper to create a level 3 data product directly from its base level 1 product each time it is requested, rather than pre-computing and storing level 2 and 3 products. This would be true if, for example the algorithm for producing the level 3 product were changing frequently and the requests for that product were infrequent.

By providing access to data indirectly through associated services, we can simplify the data access model for the user, because there's a consistent way to get to all data, regardless of the manner in which that data is created or represented. This concept is important in preserving transparency regardless of whether data is retrieved from an archive, retrieved from a traditional or object-oriented database, or computed "on-the-fly". It allows process vs. store trades to be made independently across datasets (or portions of datasets), and over time. The trade can then respond directly to economic and anticipated use constraints, and not be burdened with complex architectural implications.

Additionally, because ECS will use services to deliver data, it will be possible to reuse much of the supporting infrastructure needed for services and apply it to support the data requirements. This allows us to apply functionality developed for describing, advertising, and finding distributed services to the parallel tasks of describing, advertising, and finding distributed data. The basic "Data = Service" concept is illustrated in Figure 4.

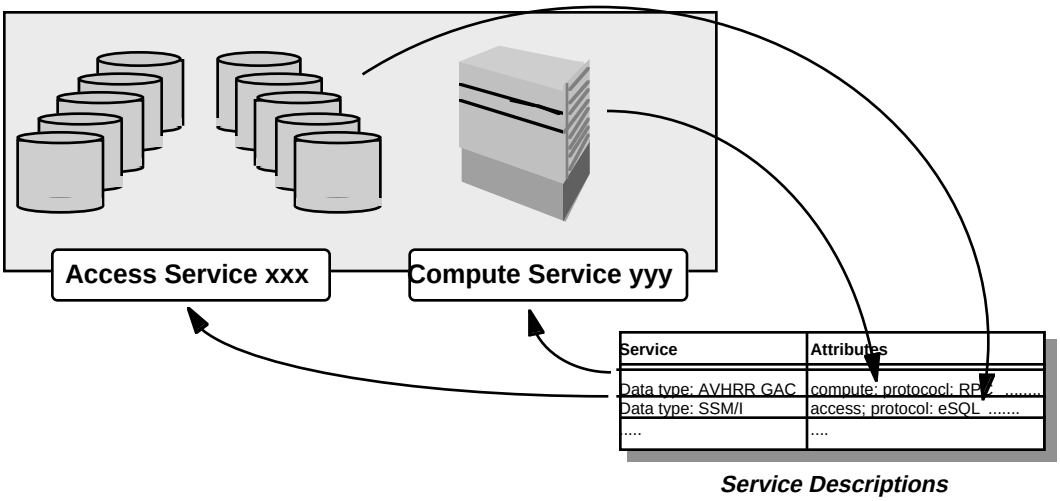

Figure 4. Data = Services

The "data = services" concept gives rise to the notion of the "Product Factory", as shown in Figure 5. The product factory is the model for product generation within ECS. The factory uses as inputs both raw instrument data and computed products. These inputs are analogous to raw and processed materials that are used to generate goods in a factory. The factory includes two primary physical "plants", the production engine and the data archive. These functions are analogous to the production assembly line and warehouse of a factory, where products are developed from raw materials, and stored as product inventory, respectively. Within this factory model, there are three significant processes involved in delivering product (information) to a user. These are:

- Production

  This process refers to the generation of a new or modified product based on raw and processed inputs, and placement in the archive (inventory).

- Retrieval

  This process refers to extracting the product from the archive, and delivering it to the requester.

- Instantiation

  This process refers to the on-the-fly generation of a product and delivery to the requester. It is analogous to the concept of just-in-time production in modern automation.

It is important to note that in both the production and instantiation steps, the production engine is programmed with either standard methods, or with user-supplied (special) methods, in an appropriate balance as determined by policy and resource constraints.

The product factory model embodies the principles of encapsulation and abstraction described earlier, by focusing at the service level on the delivery of a requested product, and by allowing the underlying implementation to provide that service in an appropriate manner. This product factory model is replicated in different forms and configurations throughout EOSDIS in support of agents we call data servers. Data servers will be discussed in more detail in the sections that follow.

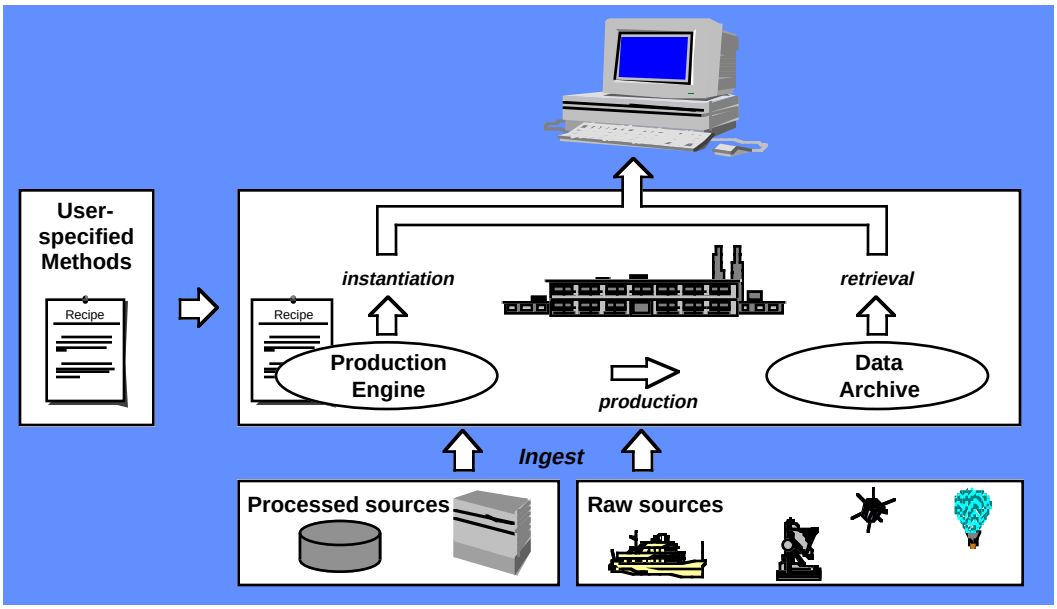

Figure 5. The Product Factory Concept

4.2.2.3. Universal Reference

A *Universal Reference* provides a universal and uniform way of referencing objects (data or service) in the ECS system. The syntax and meaning of a universal reference follow standard conventions which every component in the network recognizes and supports. It encapsulates the identifiers which are used internally at a site or by a specific service, providing a system-wide token that can be successfully exchanged for the referenced data from any location within the system. It should be stressed that the Universal Reference must not be a centrally controlled or managed referencing system or the entire foundation of the ECS architecture would be invalidated. Rather, a federated approach to the allocation of references, in which agents consult a local naming service and construct a globally valid reference, is required[6].

6 For example, Mosaic uses a common format identifier across the Internet naming domain. The reference includes a protocol specifier, and an Internet node domain name, followed by a node specific directory structure (i.e., where the file resides). In essence, the protocol and domain name components are "managed" through a globally accepted convention; the machine-specific directory structure is "managed" within the local site.

Universal references are discussed in detail in the "GCDIS / UserDIS Study" white paper, and in the Interoperability Services Architecture in section 6.1 below. Here, we highlight some key features of the universal reference concept that make it suitable for the needs of ECS. Those needs include:

- the desire to defer retrieval of information until it is needed

  By providing a "handle" to data objects, a universal reference allows search and analysis tools to provide users with summary information (metadata) about data objects, without actually sending potentially large data objects themselves. The user can examine the summary information, perhaps browsing through reduced resolution images of interest, before selecting the data to be obtained for further analysis or delivery. When the user wants to actually retrieve the full resolution imagery, he can do so through a de-referencing process, in which the data is instantiated and delivered.

- the ability to collect and use like items from multiple heterogeneous providers

  The use of universal references also allows collections of like objects to be developed and maintained irrespective of source representation of the data. Hence data objects that reside in the archive and data objects that are computed on demand can be treated equally through universal reference handles. The act of accessing the data then appears identical to the user, regardless of the underlying provider's implementation.

- the ability to easily exchange information with colleagues

  Finally, the universal reference provides a means for researchers to easily exchange information with colleagues. Rather than having to email large satellite images, for example, collaborating researchers can pass references to the data objects. The universal nature of these references guarantees the ability to instantiate or dereference the object from anywhere within the ECS system (and even beyond, assuming gateway support to convert the reference into a suitable data stream).

5. Conceptual Science Information Architecture

In this section we develop a Conceptual Science Information Architecture.[7] This architecture presents a high level organization of those aspects of the overall system architecture supporting the generation, delivery, and use of science data within EOSDIS. The conceptual architecture is based on the architecture principles and unifying concepts presented in section 4, together with a "User-Provider" model which we develop here. The conceptual architecture is presented in a layered model, with key architecture features highlighted as they relate to an envisioned operational environment.

5.1 User - Provider Model

The Conceptual Science Information Architecture is defined by a "User-Provider" model as depicted in Figure 6. This model emphasizes the fact that users and data providers are both clients of the information system's interoperability services. The model helps illustrate the fact that single sites, even single computers, may act as both users and providers within the system.

A Science Computing Facility, for example may play the role of both user and provider, accessing data used in analysis or QA, and inserting derived datasets back into the system (standard or special, depending on policy and authorization). The concept of a value-added provider, in which an individual or organization might extract information from EOSDIS, perform extensive processing and analysis, then make that analysis available to the community at large, is another example of a single site with a dual role.

5.2 Conceptual Architecture

Figure 7 presents an instantiation of the user-provider model described in the previous section. It depicts a number of physically distributed clients and service providers in a variety of configurations. These clients and service providers are connected via distributed services characterized as the Interoperability layer. We call this instantiation of the user-provider model the Conceptual Science Information Architecture.

[7] As defined earlier in the conceptual - logical - implementation hierarchy.

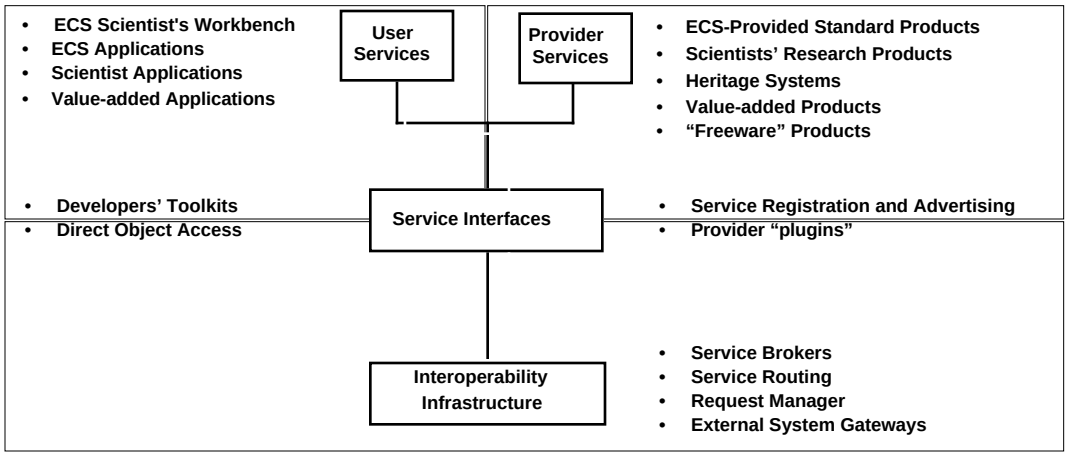

Figure 6. ECS User-Provider Model

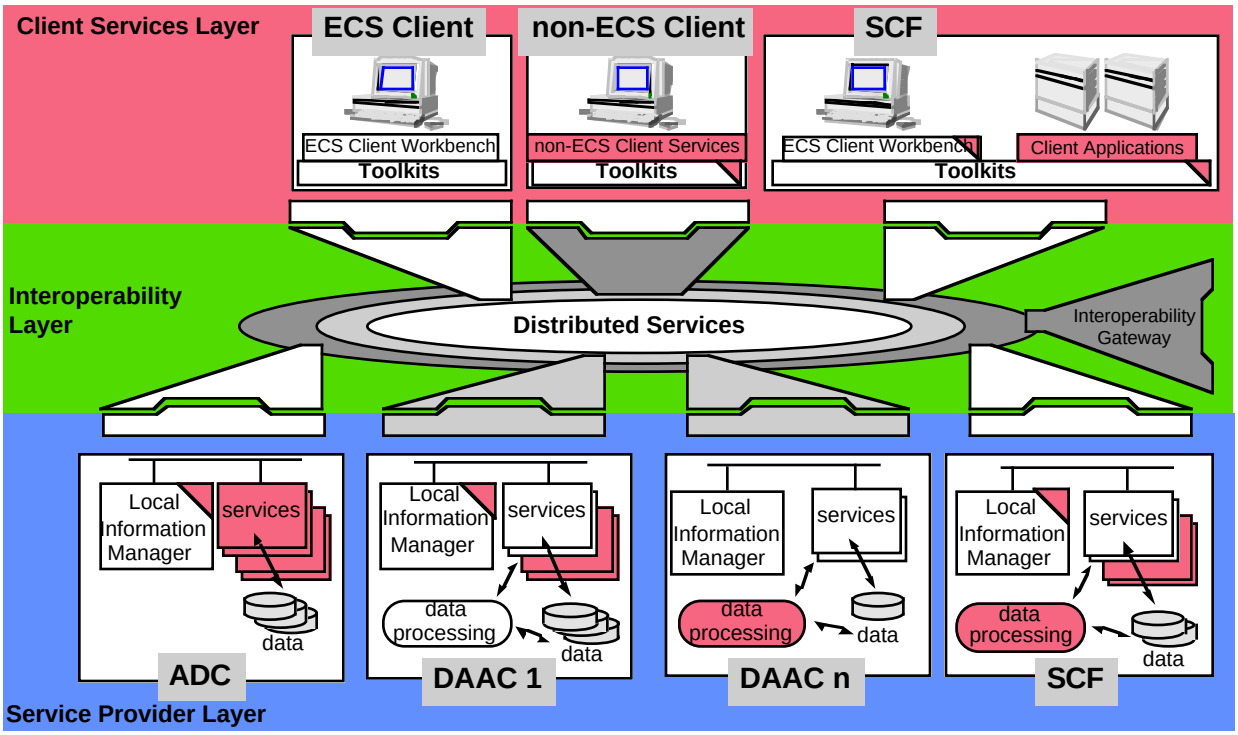

Figure 7. Conceptual Science Information Architecture

These three layers are explored in more detail in the following subsections.

5.2.1 Interoperability Layer

The interoperability layer provides the basic mechanism for connecting clients with desired service interfaces in ECS. This layer provides a buffer between clients and service providers that enables dynamic system configuration, both in the types of services provided to clients, and in the location of those services. The interoperability layer acts on the clients' behalf to provide a unified view of physically distributed and heterogeneous services. This characteristic is one of the cornerstones of the ECS conceptual architecture.

The interoperability layer actually encompasses two somewhat different levels of functionality:

- *Interoperability services* provide application level[8] interoperability components such as advertising services, request brokering services, distributed information management services, collaboration services, and subscription services.
- *Communications services* provide the core network and distributed computing services that form the foundation for the higher level interoperability services.

This distinction is important in separating higher level interoperability components from their core communications and distributed processing underpinnings. In ECS, this distinction allows for separation of application specific services from common communications services, supporting evolution of the communications and distributed computing infrastructure while minimizing the impact at the application layer (users and providers).

The communications services in ECS are based largely on COTS software, including facilities provided within the DCE framework. In a rapidly changing product environment, like that of open distributed processing, the explicit separation of interoperability and communications services allows for adoption of incremental technology components as they become available. For example, as we transition from Ethernet to FDDI to ATM for our physical network connection, we will be able to incorporate the new communications technologies and associated network and transport layer protocols without requiring rework of the higher level interoperability services.

The interoperability services are presented in more detail in the discussion of the Interoperability Services Architecture, Section 6.1. The communications services are presented in section 6.6, the Communications and Internetworking Architecture

5.2.2 Service Provider Layer

The service provider layer embodies the concepts of the "product factory", replicated in a manner that preserves heterogeneity and site autonomy. Figure 7 depicts a scenario representative of the EOSDIS environment, with DAACs, ADCs, and SCFs all functioning as data providers in this distributed architecture. These sites are unique in a number of dimensions, including: hardware and software components (computing, storage, and networks); the services

8 Here, we mean Earth Science application level functionality

provided to their user community; and the manner in which they manage facility resources. The component shading in Figure 7 is intended to highlight the site-to-site differences, affording a combination of common or standard data services as well as site-specific or site-unique services.

These services, which include both access to archived data, and data processing capabilities[9], are made available through Local Information Management (LIM) services. The LIM services function as a site-specific intermediary to data requests within a site, and support integration, formatting, and exchange of site-generated partial results with the results of other sites that may be participating in a multi-site search operation. LIM services interact with other site-specific service instantiations, including:

- vocabulary services that support translation between domain-specific vocabularies
- data services, through a unifying data server abstraction, to access specific data type services and provide appropriately formatted results
- local resource management services, that are responsible for overall product throughput and resource scheduling at a site
- services that support provider site configuration management, including processing algorithms and associated data files (e.g., instrument calibration coefficients).

The role of key provider services is presented in more detail in the discussion of the data management architecture, section 6.3, the data server architecture, section 6.4, and the data production architecture, section 6.5.

5.2.3 Client Layer

The client layer includes support for a range of site specific client environments . These environments are constructed to meet the needs of different user communities, through the development of user-specific applications. The Client Services are used to develop a host of these client applications that fall into two major classes:

- Interactive applications

  These are applications which drive some sort of user interface, and require interaction with a user. Through the client services, a variety of interactive applications will develop over time, suited to meet the needs of a variety of users. Some of these applications will be developed within ECS by the ECS contractor; others will be developed by researchers throughout the EOS community; still others might be provided through COTS products or value-added providers. We envision an environment capable of eventually supporting:

  - GUI-based data access tools supporting system navigation in a number of ways, including:
    - › directory information
    - › inventory information

9 through the product factory concept

    - › incremental search
    - › results review, including various data “abstractions”, such as browse data
  - Data analysis and visualization tools, including animation capabilities
  - Hypertext-based electronic journal applications
  - Collaboration tools that support shared windowing environments, electronic white boards, textual and audio “chat” sessions, and video teleconferencing
- Non-interactive (processing) applications

  These applications are algorithmic transformations of data. This class of applications typically require no user interaction once running, although there may be some configuration required prior to execution (determining input and output datasets, etc.). Examples of typical applications in this class include:

  - automated QC applications
  - subscription agents (applications that pick up information “delivered” to a user via subscription)
  - data analysis and transformation applications (i.e., applications that analyze data products to produce derived or modified products)
  - data “mining” applications capable of monitoring new developments in the information repository, and notifying users of items of interest

Both interactive and non-interactive applications are developed on top of ECS-provided toolkits, perhaps with user extensions. The thought here is that ECS will develop a set of fundamental toolkit capabilities, but that these might be extended to provider site-specific capabilities to higher level applications. Such extensions might subsequently be shared within the larger science community, creating, for example interoperable non-ECS clients capable of accessing portions of the EOSDIS data repository. For example, users might develop a set of tools to support multi-media annotations to in situ data measurements. An SCF could develop the necessary services to support capture (“ingest”) and playback of audio or video annotations through a set of extensions to the basic data management services, using fundamental services as a foundation for development.

The client services are presented in more detail in the discussion of the client architecture, section 6.2.

5.3 Conceptual Architecture Characteristics

In this section, we explore some of the key characteristics of the Science Information conceptual architecture. This includes a discussion of key architectural features that embody “logical distribution”, discussion of a user perspective of the system, and a discussion about the scope of activities envisioned under ECS, and within EOSDIS at large.

5.3.1 Logical Distribution

The distributed service architecture described above supports a logically distributed system architecture characterized by the following key features:

- Heterogeneity
- Autonomy
- Location transparency

These features are discussed in more detail in the following subsections.

Heterogeneity

The distributed service architecture supports heterogeneity among the various components at all three layers, with differences across clients and providers being most obvious. Sites within EOSDIS will be different from one another in many ways, including:

- different physical system components

  This includes possible differences in the number, type, and configuration of hardware platforms, host operating systems, and local area network technologies and topology.

- different arrays of data products and data types

  In addition to differences associated with the mix of standard products, there may be significant differences in site-unique services offered. An SCF, for example, might offer specialized data products that it computes based on new algorithms embodying an investigator's recent research. These products might be very limited in spatial and temporal coverage, owing to their exploratory nature. A DAAC, on the other hand, may offer a wide range of standard data products (and perhaps multiple versions of products at certain points in time). DAACs might also offer up "DAAC unique" products to its primary user community based on common analysis requests. Yet another site in EOSDIS may act as a hypertext document repository for an electronically published journal. This site might provide full-text indexed hypertext documents as a service to ECS clients.

- different client applications and tools

  Client sites may differ greatly in the environment they provide to their users. While some sites might run only a standard ECS client application supporting data search and order (and browse and display for appropriately equipped workstations), others might include a wide variety of analysis and visualization tools. These tools may be developed using a combination of ECS-provided and user-developed code.

Autonomy

While the concept of heterogeneity is concerned with supporting different configurations and service profiles from site to site, autonomy is the property which enables each site to "govern" itself. In EOSDIS, some sites will most likely be more autonomous than others, due to the need for some sites to shoulder fundamental responsibilities in carrying out mission critical objectives

(e.g., DAACs have substantial operational responsibilities). But, in general, the distributed architecture is aimed at supporting sites which manage their own realms -- users, resources, and processes. The basic concept is that an autonomously managed site "plugs in" appropriate resources, users, and processes (computational "methods") to the EOSDIS system as a whole. These resources, users, and processes then become part of the ECS web, governed by an appropriate set of overriding resource management and authority policies.

Location Transparency

The distributed services that are part of the interoperability layer support the notion of *location transparency*. Location transparency means that, from a functional standpoint, it doesn't matter where a service physically resides in the system – services are available universally throughout the system regardless of the location of their physical instantiation.

This location independent view of the system protects users from changes in system configuration over time, and allows services to be migrated from location to location based on economics, operations, and other resource considerations. Migration of services could occur in response to changes in resource needs, changes in input data sources (and hence proximity of data sources), or configuration changes required by malfunction or component obsolescence. In the most dynamic example, the system could even decide to migrate services to data for the execution of a single or a few operations. This might be done as part of an optimization approach based on the location and size of data, and the complexity of computational operations.

This location transparency is provided by the *dynamic service binding* capabilities of the interoperability layer – i.e., relevant services are found at time of need, rather than through predetermined configuration control. In addition to allowing for service migration, dynamic binding also supports the development of new services. This supports an evolutionary system development approach, in which core user services can be provided initially, and new services added to the system as they are developed.

Note that, while the system supports evolutionary service development, whether or not different clients can take advantage of that service is another matter. For example, a client tool developed to perform text searches over a repository of journal articles may not have direct access to services which deliver rasterized satellite imagery. However, the ability to access that image service is afforded to clients of the system, regardless of their physical location. (In actuality, there may be access restrictions imposed by the system to enforce resource allocation requirements, or to enforce data protection policies. This issue is discussed in more detail below).

Logical distribution and Physical Sites

The distributed service-oriented architecture presented here allows us to develop classes of services that transcend physical architectural boundaries. In the end, however, these services are manifested at physical computational sites. The service decomposition described in the layers of the conceptual architecture allows us to configure physical sites as necessary to meet the needs of the local community. Hence sites actually include a variety of components from the three layers presented earlier: Client, Provider, and Interoperability services. This allows a single physical site to act as data provider, a user, or, as is the case with most SCFs, both a provider and a user.

A single site's computing, storage, and communications resources can therefore be used to support both data production and data consumption needs, under the control of a local authority.

5.3.2 User Perspective

The ECS system will support a tremendous range of users, with a diverse set of needs. It is impractical to provide one set of tools and one view of the system that can meet the needs of all users. Such an approach provides an overly complicated interface to users with modest information access needs, and often compromises the capabilities delivered to "power users".

Earth science researchers, policy makers, teachers, students at a variety of educational levels, and system administrators and operators are all users of the system, each with different needs in accessing information. A researcher, for example, may need access to alternative data products for a specific geophysical quantity in order to explore the impact of new processing methods on her research. Or she may even need access to the raw sensor data to explore the impact of new atmospheric correction algorithms. An eighth grade science teacher in Maryland, on the other hand, might simply require access to local AVHRR data to explore local weather patterns with his class.

Additionally, the ECS system has a limited amount of resources (processing, storage, distribution bandwidth) to offer to the community. It will be important to manage those resources in an effective manner to prioritize use of system resources and ensure the completion mission critical operations.

As described earlier, the architectural foundation for ECS provides a single logical information management view of this physically distributed, heterogeneous system. That single, global view will actually be partitioned into a number of overlapping subviews through a combination of *user tools* and *user realms.*

A user realm is determined by the capabilities and privileges granted to a user by the system. It is determined by a user profile, which may include both system-imposed and administrator- or self-imposed restrictions on what the user sees and what the user can do within the system.

For example, an earth science researcher's realm might include access to level 1 data and processing algorithms for one or more instruments key to his fundamental research. He might also have the ability to request the generation of special data products, either routinely (as resources permit), or on an as-needed basis.

A user's view of the system is also determined by the tools he uses. A user might establish a profile that automatically focuses the system's search capabilities to areas he frequently uses. Hence a researcher who only needs selected data products from a specific DAAC might establish a profile in a search tool that restricts his view of the world to the subset of interest.

5.3.3 EOSDIS scope

The conceptual architecture presented in this document attempts to leave open as many evolutionary paths as possible to support system extension and growth. In providing a heterogeneous, distributed, service-oriented architecture, it allows for the development of

numerous system capabilities which may not directly relate to primary ECS objectives. As such, it is important to distinguish between capabilities and functions that the architecture *allows*, and those that it *provides.*

The services, functions, and tools provided by the ECS contract will assuredly be a subset of those that become available within the EOSDIS community. Users and value-added providers will be able to develop additional system and component capabilities that find their way into the EOSDIS community. These activities may manifest themselves as enhanced client applications, expanded toolkits, DAAC-unique services, SCF-provided data analysis, and the like. The delineation between ECS provided tools and services, and value-added tools and services will become clearer as higher level system requirements are refined and the System Design Review (SDR) approaches.

6. ECS Sub-Architectures

We now apply the architectural philosophy developed in section 4, and the conceptual architecture presented in section 5, to develop a series of logical sub-architectures. These architectures provide a set of system perspectives, each focused on a different aspect of system functionality. The sub-architectures allow for decomposition of the system along a number of different lines, and to different levels of detail, providing complementary models of the system useful in further analysis and design.

Unlike a functional decomposition, these models may in fact overlap or subsume one another in their perspectives – i.e., they are not strict, equivalent-level subsets of the overall architecture. Rather, they provide a framework in which to explore key issues associated with an aspect of the system. However, together, these models are intended to provide adequate system coverage. Where these perspectives overlap, they provide for cross-checking among the various perspectives of the architecture, supporting completeness and consistency checking of the overall system architecture.

These perspectives may include components in each of the layers of the conceptual architecture, illustrating how the different levels of components in the system come together to support key system functions. For example, in the data management architecture described below, the query interface needed to search for data belongs in the user domain; the infrastructure needed to route search requests to appropriate sites belongs in the interoperability domain; and the data management and access services needed to execute search requests against the data stores belong in the provider domain.

The sub-architectures described in the following sections focus on issues related to the services that support users or client applications.[10] The service-level model is augmented by user and data modeling activities aimed at characterizing current and anticipated uses of the system, and anticipated dataset needs. A parallel object modeling activity is underway to fully develop the objects- and services- based approach described earlier. As such, this document represents work in progress in the object modeling and service decomposition activities. These activities are all part of the ongoing system design effort, and are being documented in a series of internal working papers. The results of these activities will be part of the System Design Specification and will be presented at the System Design Review.

Scope

At the level of the logical architecture decomposition, we choose to explore the upper portions of a system service hierarchy as depicted in Figure 8.

[10] In this context, we sometimes use the term "service" loosely, to describe functionality that the system makes available to its clients, perhaps through multiple service interfaces.

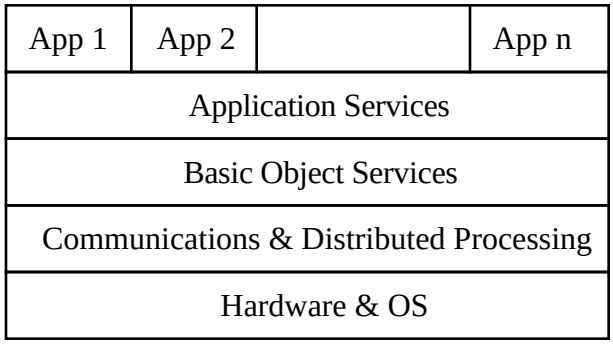

Figure 8. System Service Hierarchy

The hierarchy shown in Figure 8 extends down to the level of the hardware and operating system in order to provide appropriate context for this discussion. However, the service model concentrates on the relationship between the top two layers: the applications themselves and the supporting application services layer. In this context, applications include both users and providers, and may include completely automated applications, or human-in-the-loop applications. The Application Services layer includes specific services developed to support client, service provider, and interoperability functionality. This layer in fact extends to a service network -- a hierarchy of more fundamental services called upon by one another to provide the functionality of the logical architecture. A simple network of services is depicted in Figure 9.

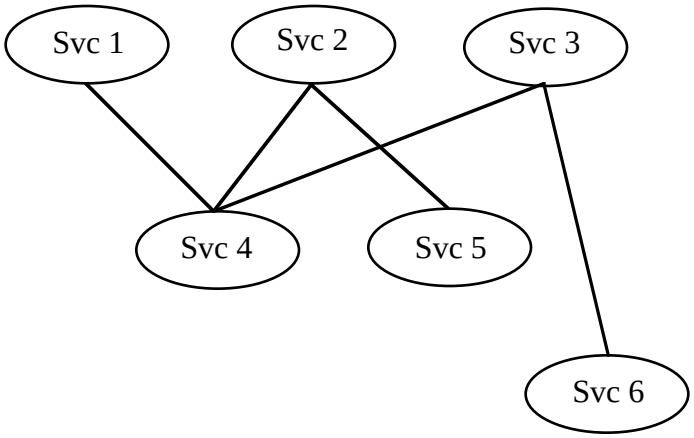

Figure 9. Application Service Network

The Basic Object Services layer of the System Service Hierarchy includes fundamental system services on which the application services can be developed. This layer might include commercial distributed object management services (e.g., as outlined in the OMG Object Services Architecture “Categories of Object Services”) to support fundamental service

advertising (export), use (import), and acquisition (brokering). Additional services present at this layer, and on which higher level application services would be developed, might include conventional and/or object-oriented database (OODB) services.

The Communications and Distributed Processing layer includes lower level fundamental communications and distributed computing support, such as network protocol stacks, RPC, naming services, network authentication and security services, etc. Some of this support will likely be provided by OSF's Distributed Computing Environment (DCE).

Organization and Format

Throughout the remainder of this document, the Basic Object Services and Communications and Distributed Processing layers are discussed only to the extent that they directly support functionality at the application services level. Additional details of the basic object services layer, plus the lower layers of the System Service Hierarchy, will be expanded in follow on analysis and design efforts.

In the following subsections, we define and discuss the following sub-architectures:

- Interoperability Services Architecture
- Client Architecture
- Data Management Architecture
- Data Server Architecture
- Data Processing Architecture
- Communications and Interconnection Architecture
- System Management Architecture
- Flight Operations Architecture

The first five of these sub-architectures are developed within this white paper. The remaining three (Communications and Interconnection, System Management, and Flight Operations) are sub-architectures being defined outside of the Science Data Processing Segment (with appropriate collaboration, of course). In this paper, we attempt to characterize them at a fairly high level and to describe the interactions with SDPS components.

The subsections below are organized as follows. First, appropriate background material is presented, explaining the scope of the logical architecture being described, and tying back to some of the conceptual architectural discussions. Next, key concepts and models of the logical architecture are presented, including a high-level description of the fundamental service classes within the architecture. Finally, there is a discussion of some key implications and issues associated with the components of the logical architecture, including discussions of functionality, and issues of risk and architecture evolvability.

6.1 Interoperability Services Architecture

The Science Information Architecture is based on an augmented client-server model as described earlier. The augmentation is provided by the interoperability layer, ensuring the flexibility and scalability required to develop an information system as aggressive and complex as ECS. Some of the challenges in designing and implementing the conceptual architecture as put forth in section 5 include:

- clients must be able to determine what services are available on the network and where;
- the compatibility of the requirements of a client with the interface offered by a service must be determined;
- requests must be routed from clients to services and responses must be transported back;
- clients must be notified of changes in the configuration of services;
- a method is needed by which objects (e.g., data) which are managed by services can be referenced throughout the network; and
- services must be monitored and administered to track overall system loading, balance work loads, and manage system availability and reliability.

In modern communications networks, many of these functions are assigned to network applications such as the distribution and network management services found in OSF DCE and DME. However, these services do not deal with the semantics of the application environment itself. For example, DCE cannot determine which database can meet a client's requirement for a given earth science data product and whether the data is available in the requested format.

Therefore, ECS has added an ECS-specific application layer on top of the traditional network applications layer. That layer is called the ECS Interoperability Services layer. It augments the network applications provided by the communications infrastructure, addressing the above issues at the level of application semantics[11].

6.1.1 Concepts and Reference Model

ECS communications services[12] provide the basic mechanisms for connecting a client with a specific server interface. However, they cannot provide the facilities needed to specify and resolve client requirements which are specified at the level of application semantics (e.g., in earth science terms). ECS therefore extends the basic client server model to include several additional services. They are called interoperability services. Interoperability services provide a collection of functions which ensure that requests are routed to appropriate services. They also provide a general framework for the interaction of clients and servers, e.g., for the exchange of data. In doing so, they support some of the key features of heterogeneity, autonomy, and service evolution as described earlier.

---

11 The relationship between ECS interoperability services and the communications architecture is described in the Communications and Interconnection Architecture, section 6.6.

12 as described in the Communications and Interconnection Architecture, below.

Examples of the three most important interoperability services are shown in Figure 10. They are the:

- ECS Advertising Service;
- ECS Request Brokering Service; and
- ECS Universal Naming Service;

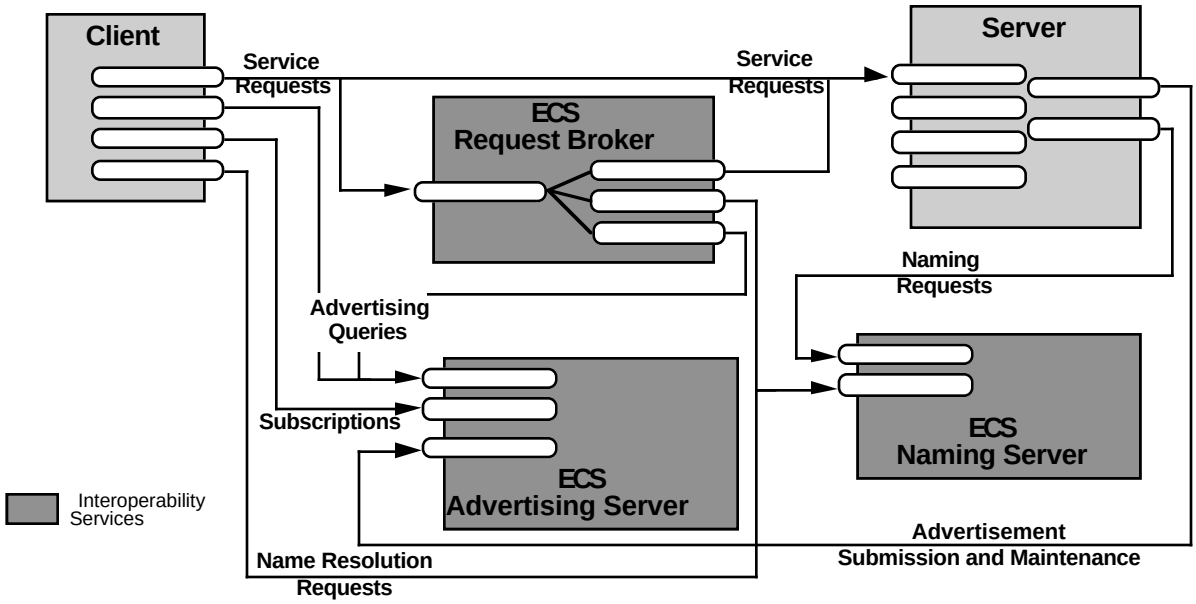

Figure 10. ECS Client Server Reference Model

The advertising service manages information about the services available in the ECS network. The information is called services advertisements and is supplied by the servers. The ECS architecture associates a definition of an advertisement with each class of ECS service, and specifies an interface and protocol for submitting and maintaining advertisements. The advertising service also provides interfaces for querying this information which are available to any client.

The request brokering services accepts service requests from ECS clients and locates appropriate servers. It uses the query interface of the advertising service to match the service requirements described in the request against available advertisements. It can route the service request, or return the information regarding matching servers back to the client. Clients could also query the advertising service directly, if they prefer. However, using a request broker decouples clients from advertising interfaces and the schema of advertisements, and is therefore the preferred approach.

The pointers which clients receive are called *Universal References*. The use of a universal reference in supporting a distributed client-server transaction is depicted in Figure 11. A universal reference can point to a service, a specific service interface, and any object managed by a service. The purpose of a universal reference is to make the methods by which services identify objects internally transparent to the clients. Rather than providing this internal identifier, a service converts it into a universal reference and transmit that reference to a client. All universal references have the same external appearances and can be managed by clients in a uniform way. However, when presented to the service which originally provided the reference, the service will be able to reconstruct the internally used identifier.

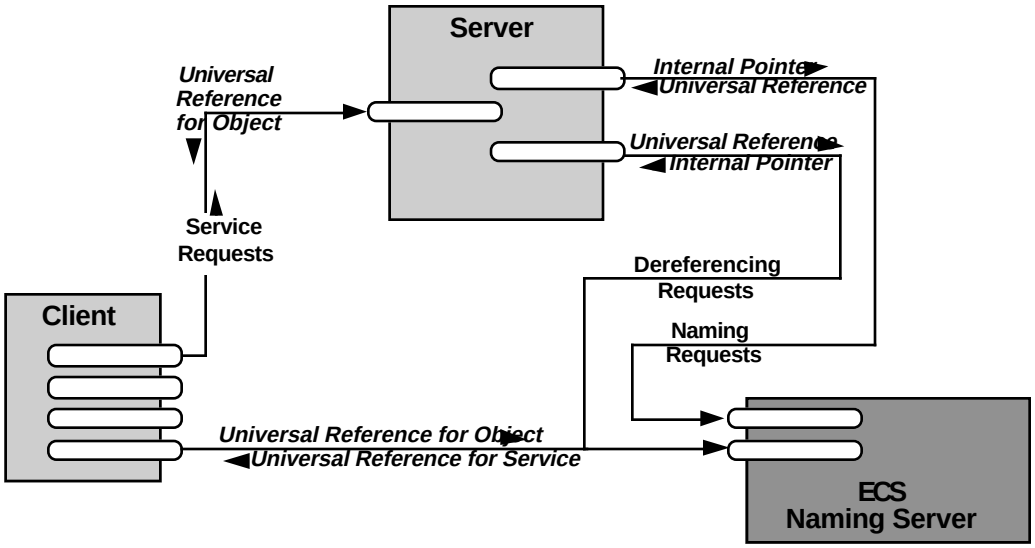

Figure 11. Universal Naming Service

The *Universal Naming Services* provide a mechanism for obtaining and resolving universal references. Clients can call a universal naming server to convert an internal reference into a universal one, or to dereference a universal reference (e.g., in order to identify the server which originally supplied the reference).

6.1.2 Implications and Issues

In the following discussion, we explore a number of implications and issues surrounding the Interoperability Services Architecture.

Use of Advertising and Request Broker Services

Clients can use the ECS interoperability services to locate services, transmit and manage requests, obtain request results and receive notification of service changes. However, ECS clients are free to limit their use of such services according to their needs, for example in the interest of improved performance. For example, a client (through a request broker) might

employ an advertising service to determine the availability and source of satellite observation data. The advertising service would return a universal reference to the provider of this data.

Once a client has obtained a pointer to a server, it can send requests directly to that server and avoid further request brokering overhead. However, the client could also continue to use the request broker to safeguard his software against possible changes in service configurations. For example, a data set might be moved between sites. A client using the request broker would have its requests automatically routed to the new location.

Alternatively, the client might be notified of this change (by the subscription service); or might use the advertising service again after a data access request failed because the usual provider no longer offers the requested data. In the meantime, however, the client would converse with the service which provides the data directly, without using the ECS interoperability services (i.e., using only the network applications).

Advertising Subscriptions

The advertising service also offers an interface for subscribing to changes in advertisements. The interface is available to any ECS client. Request brokers might cache advertising information; they could use the subscription mechanism to maintain the cache. Clients which send requests directly to servers located via the request broker could use the subscription mechanism to update that information when the configuration of ECS services changes.

Other Interoperability Rules and Services

The subscription interfaces offered by the advertising service are an instance of a general interoperability function which the ECS architecture provides and specifies. There may be other types of services for which a subscription interface might be useful. For example, data management servers may offer to inform clients of changes in their data holdings (e.g., the availability of new data products). As part of the ECS interoperability rules, the interoperability architecture specifies a number of such functions and interfaces, including the following:

- subscribing to services;
- obtaining extended service description;
- obtaining help services;
- obtaining and negotiating service profiles;
- managing ongoing service requests;
- obtaining and negotiating format profiles for data exchanges;
- asking for costing information;
- obtaining dictionary and vocabulary information;
- supporting authentication and acknowledgments; and
- supporting system and network management.

These interfaces and functions are described more fully in the Interoperability Services Architecture Working Paper.

Ensuring interoperability

ECS interoperability services will only operate successfully if all other ECS services meet and implement their requirements. The following are examples:

- The catalogue managed by the advertising service must be populated with the descriptions of the available services. Services must cooperate by providing and maintaining these descriptions.
- When a service provides a reference to one of the objects its manages, it must invoke the universal naming service to convert the service-internal reference into one that can be used by clients throughout the network.
- Network and system management applications expect ECS services to support interfaces via which they provide status information and work statistics and accept requests for the performance of administrative functions.

The ECS interoperability services architecture, therefore, also defines a set of interoperability rules. They specify functions and interfaces which all ECS services must provide. They also include a set of optional functions and interfaces which services can elect to provide, and a standard method (called a service profile) by which services define which of these options they support. The ECS client-server interactions include mechanisms which allow a client to request a service profile for a given interaction, and enable a service to agree to or refuse that profile request.

There are also a number of issues associated with the design of the interoperability services. The issues must also be considered in ensuring interoperability across the system.

- Services need to support interoperability by providing and maintaining advertisements. Policy guidelines controlling the submission, withdrawal, and invalidation of advertisements still need to be established and agreed upon among the architects and designers.
- The architecture effort still needs to develop standards (or at least requirements) for service advertisements, service descriptions, help services, and a number of other data objects which are exchanged by interoperability protocols.

There are a number of more subtle issues associated with the implementation of advertising services within ECS, including:

- synchronization of distributed components of the advertising service.
- advertising service “proxies”, where a local advertising service contacts another advertising service on the requesters behalf for advertising information which it does not itself posses.
- management of advertisements, such that advertisements are guaranteed to be current, accurate (consistent with the advertised service), and authorized.

- the development of standard procedures for dealing with advertising service errors
- uniformity across multiple physical implementations (e.g., different machines, operating systems, and service representations)

Interoperability and Communications Services

The relationship between the interoperability architecture described here, and the underlying network communications support, is an issue requiring some additional analysis. It is addressed briefly here, and in more detail in the communications and internetworking architecture discussion of section 6.6.

In the ECS model of system interoperation and interconnection, the interoperability services are network applications and form a portion of the OSI applications layer. They share this layer with the network application services (such as remote procedure call and network directory services), but logically, the interoperability sit atop the network service services, That is, they use the network services for their own communications. For example, the request broker might use remote procedure calls to interact with the advertising service; and the universal references provided by the ECS universal naming service might contain network symbolic names defined in the network directory.

However, the line dividing the two layers is not clear cut and there is a potential for functional overlap between some interoperability services and some network services. For example, the OMG network model includes a common object request broker which provides some of the functions assigned to the ECS request broker. The reference model for open distributed processing includes a trading service which combines ECS request broker and advertising service functionality.

For this reason, some of the ECS interoperability services might be implemented by the ECS Communications and System Management Segment (CSMS), for example, as extensions of services already existing in the network applications layer; or they might be implemented as applications in a layer above the communications services. Until this issue is resolved, the interoperability architecture will continue to treat all interoperability services as if they were implemented as SDPS applications, above the communications services layer.

6.2 Client Architecture

The Client Architecture focuses on aspects of the system that affect the manner in which clients access the services available to them. In this context, a client refers either to a human user accessing the system through an interactive workstation or terminal, or to a computer-based process accessing a service. In the Client Architecture, therefore, "clients" will exist in both the user and provider layers described previously, with provider clients being involved in managing the servers and resources involved.

The service interfaces seek to represent the ECS as a collection of online services. The physical topology of those services is transparent to the user in order to support dynamic configuration and migration of services without affecting user interaction. As described earlier in the conceptual architecture discussion, it is this fundamental concept of location transparency that supports the logical distribution of the ECS.

Additionally, the client architecture relies on a dynamic service description and advertising capability to allow evolution of services. The fundamental concept of dynamic service binding supports, among other things, the evolution of a data provider's capabilities, and the creation of new data providers as the opportunity and need arises.

6.2.1 Concepts and Reference Model

Service Interfaces

The functionality described above is delivered using three types of service interfaces that exist within the Client Architecture:

- general interfaces
- specialized interfaces
- object interfaces

Each of these types supports both user- and process-based clients. The characteristics of each of these types are considered below.

General Interfaces

General interfaces are ECS-developed interfaces that provide compatible access with ECS-provided services. Different instantiations of these interfaces are used for different service contexts. For example, a user searching the entire collection of EOSDIS data sets using common query parameters will interact with a different interface than a user employing data set specific attributes to search within a single data set. The concept of multiple interface instantiations is shown in Figure 12.

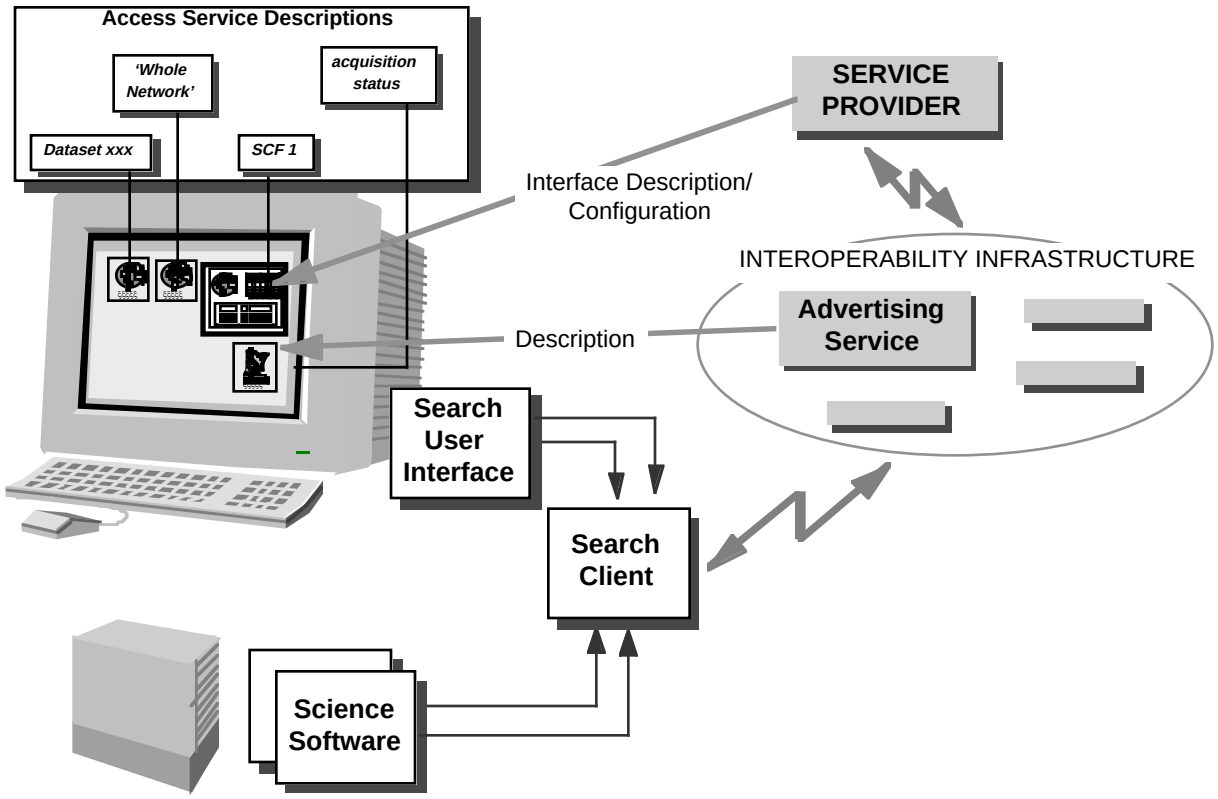

Figure 12. The General Service Interface Concept

The diagram shows three instantiations of the search interface, each with a different "view" into the ECS system. Two of the interface instantiations are shown as icons; the third interface (to SCF 1) has been activated. The support for an iconic view is announced by the advertising service and the configuration of the interface is delivered by the service provider in order to maintain compatibility between the service and interface configurations. The figure shows a similar multiple instantiation interface scenario for process-based clients, such as science software algorithms.

The general service interfaces include functions to:

- search for all types of data and metadata
- visualize, transfer and manipulate data
- place acquisition requests and review acquisition status
- place subscription requests and collaborate with other users

Specialized Interfaces

Specialized interfaces provide functionality which cannot be achieved through dynamic configuration of one of the general interfaces as described above. It is envisaged that these

interfaces would be provided through dynamic linking of software "objects". Instead of a configuration description being transferred by the provider as for the general service interfaces, the entire software object would be sent to the client workbench and dynamically linked into the client's environment.

The specialized interfaces will not be developed as part of the ECS program, though the concept of a client workbench which provides general interfaces must also support specialized interfaces offered by service providers through the dynamic linking process. It is envisaged that specialized interfaces will initially be developed for SCF and DAAC unique services, though this approach could also be employed by users wishing to develop alternative interfaces to common DAAC services.

Object Interfaces

Object interfaces provide an efficient means of binding system objects and services together to create new system "objects", perhaps with customized behavior. For example, an inventory query may "return" a set of results; this results set is presented to the user as an icon. When the user "activates" the icon (by double clicking on it, for example), he is actually triggering access service links within that results object. The information associated with that object (e.g., the actual images comprising the results set) may physically exist anywhere in the system, and in fact, may be *distributed* throughout the system. In the case of the inventory search, "activating" the results object could invoke a service to visualize the results or to browse the granules identified in the inventory search. The object services concept is depicted in Figure 13.

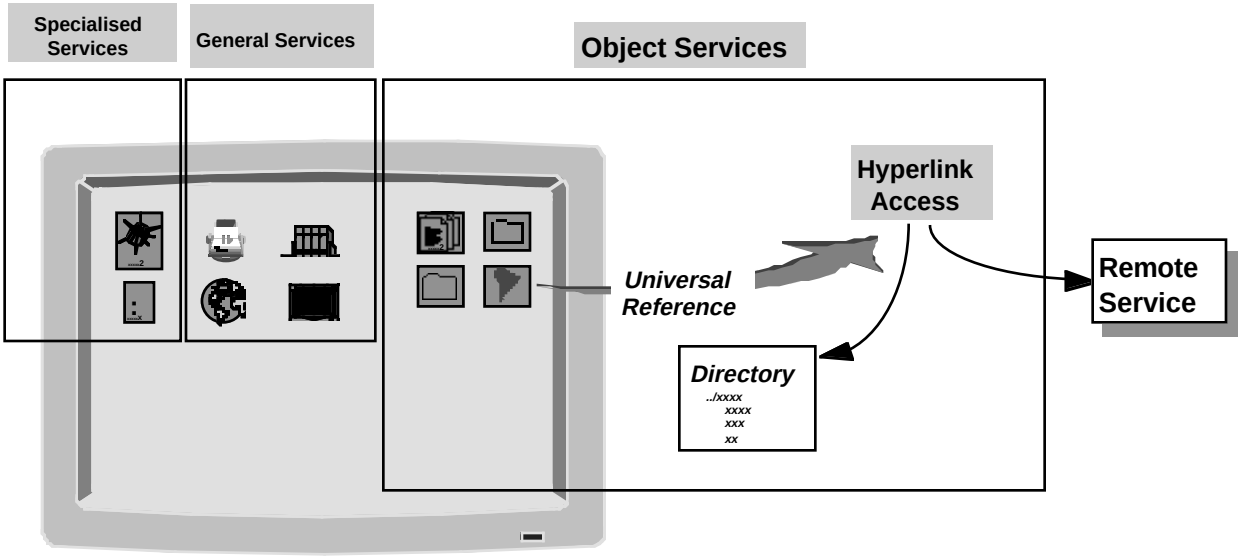

Figure 13. The Object Interface Concept

The object interface concept provides an efficient means of capturing all the information required to access a data object, without the need to explicitly replicate the data itself. This is useful, for

example, when users wish to share results. The object that represents the data can be passed or shared between users without explicitly copying the data. Recipients can then independently access the object and the services it offers at a later time. Additionally, users could refine these objects with additional services, for example to support access via custom visualization tools.

6.2.2 Implications and Issues

Composite Services

The combination of general, specialized, and object interfaces as presented above provides a very powerful mechanism for developing and extending the set of user-level services. One example of this is the development of an information subscription service, as depicted in Figure 14. A subscription service is offered by a provider, and allows users to define standing search requests that the provider will execute periodically on the user's behalf. Users are typically interested in finding out when new information sources are available on a specified topic, and would be notified of such an event in the form of an object interface as described above.

The figure also illustrates how the same interface could be used in conjunction with an advertising service to implement a subscription service notifying users when new services are registered. The extent and sophistication of the subscription services offered within ECS still requires further definition, however we believe that some fundamental subscriptions will be offered (i.e., electronic journals). Users and value-added providers should similarly be able to develop subscription services, as long as those services stay within the resource envelope afforded their users.

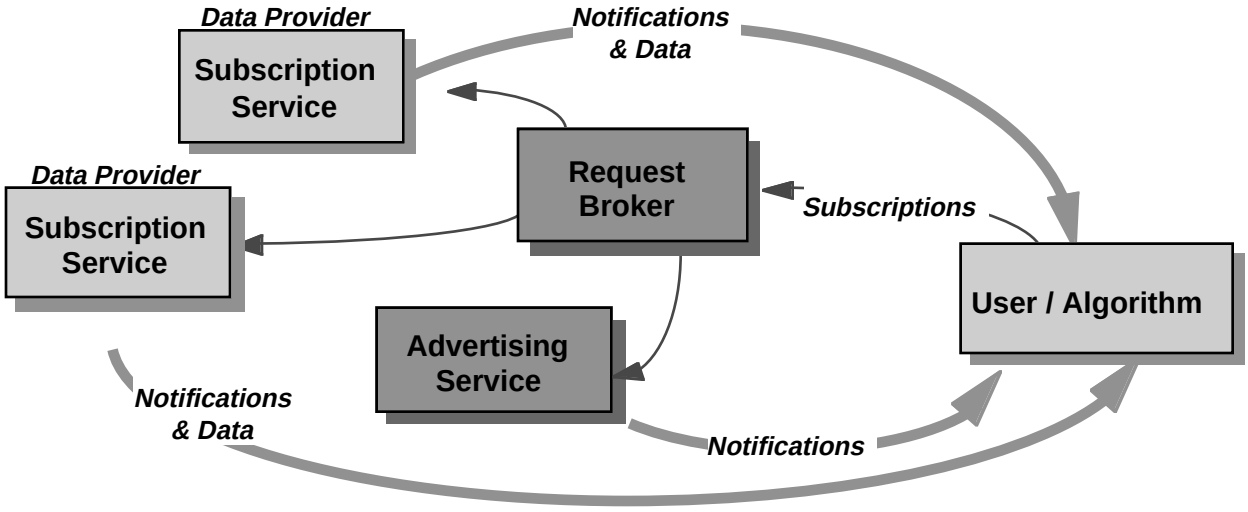

Figure 14. Subscription Service Concept

6.3 Data Management Architecture

In this section, we explore the data management architecture. This architecture focuses on key issues and challenges associated with the management of information in ECS, as derived from the mandate presented in Section 4:

- support product "publishing" and "access"
- provide a seamless view of all data
- support an extended and extensible set of providers
- build upon heterogeneous, autonomous system components

The architecture has been developed through consideration of key user needs for extensive information retrieval capabilities. These capabilities require increasingly sophisticated intersite search services that accommodate complex, coincidence searching of data across all disciplines. As new patterns and features are discovered in the data, more integrated content-based search needs will become more widespread and will begin to compete with standard production services for processing system resources.

Many of the users' stated search needs are quite aggressive, owing to the anticipated size and complexity of the data repository. It is unlikely that early ECS releases would be able to accommodate the more demanding requirements without undue schedule and technical risk. Hence one of the key concerns of the data management architecture is to provide a framework which permits the evolutionary progression from the initial data management implementation towards the envisioned long term solution.

In the near term, the ECS team will establish the data management and access infrastructure and basic architectural components while adapting Version 0 components to work within that infrastructure. Figure 15 provides a high level view of that infrastructure, cast within the layered model of the conceptual architecture. It provides three levels at which data access requests can be processed: at the intersite level by a *Distributed Information Manager*; at the site level by a *Local Information Manager*; and at the data set or data type level by a *Data Server*. These services may optionally be accessed through the request broker and service advertisement mechanism described in the interoperability services architecture.

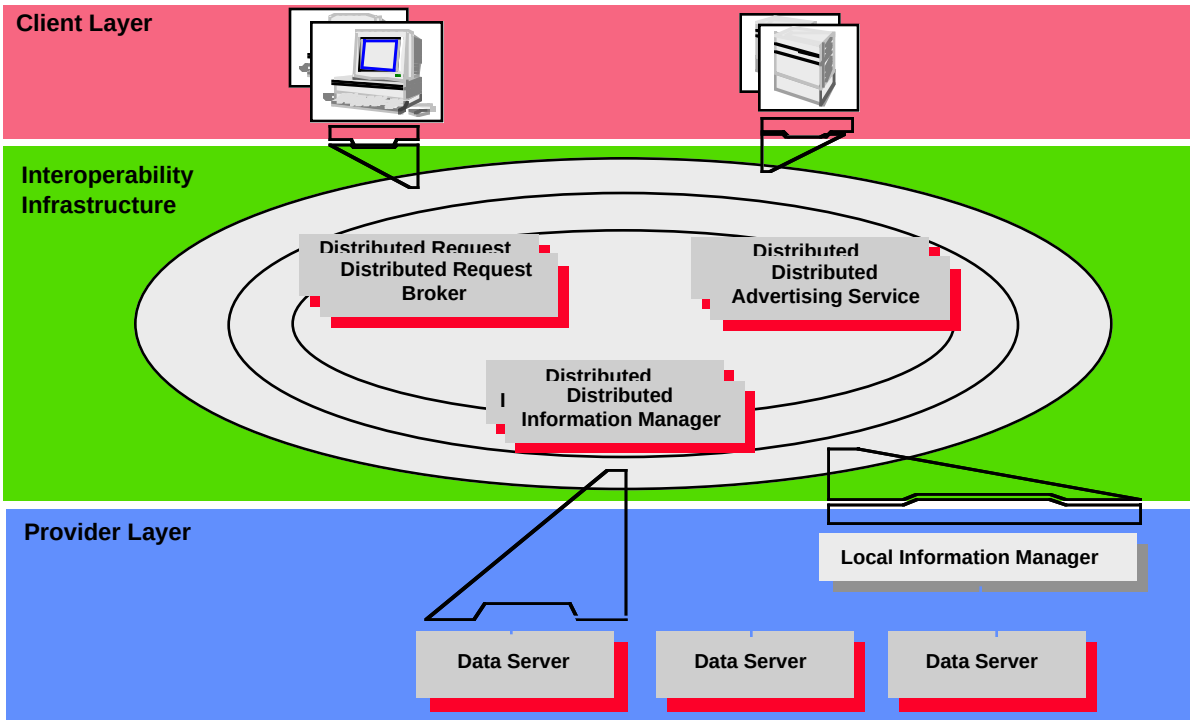

Figure 15. Top-level Data Management Architecture

The data management architecture accommodates evolution by decoupling key aspects of data management and allocating them to separate architectural levels; and by defining the available search and access capabilities in a schema which can be expanded as capabilities grow. Finally, Earth Science data types and their associated services can themselves evolve by continued development of the data servers.

6.3.1. Concepts and Reference Model

This section provides a reference model for the data management architecture. It identifies the key components, interfaces, and data objects in this architecture and defines their roles. The model is shown in Figure 16 and is a variant of the ISO Data Management Reference Model (ISO 10032:1994). The variation is in the local (site) data management. The ISO reference model does not provide a structure for local data management. ECS defines local data management as being provided by a local information manager and a collection of data servers.

The figure shows the following services and requests:

- *Client:* a program requesting data management services. A client can be an application program, such as a science algorithm, or a user interface, for example, an interface for formulating database queries and displaying query results. The client may use a data dictionary or vocabulary to assist in the formulation of database requests.

- *Distributed Information Manager (DIM):* an instance of a service which is capable of executing requests requiring access to multiple sites. The DIM acts as an agent for the client: after the DIM accepts a distributed query or access request, it assumes responsibility for its execution and for compilation of the result; the client can disconnect from the DIM, reconnect at a later point in order to determine the status of a query, obtain partial results, or cancel the query. The DIM uses a Distributed Schema to determine how a request should be executed.

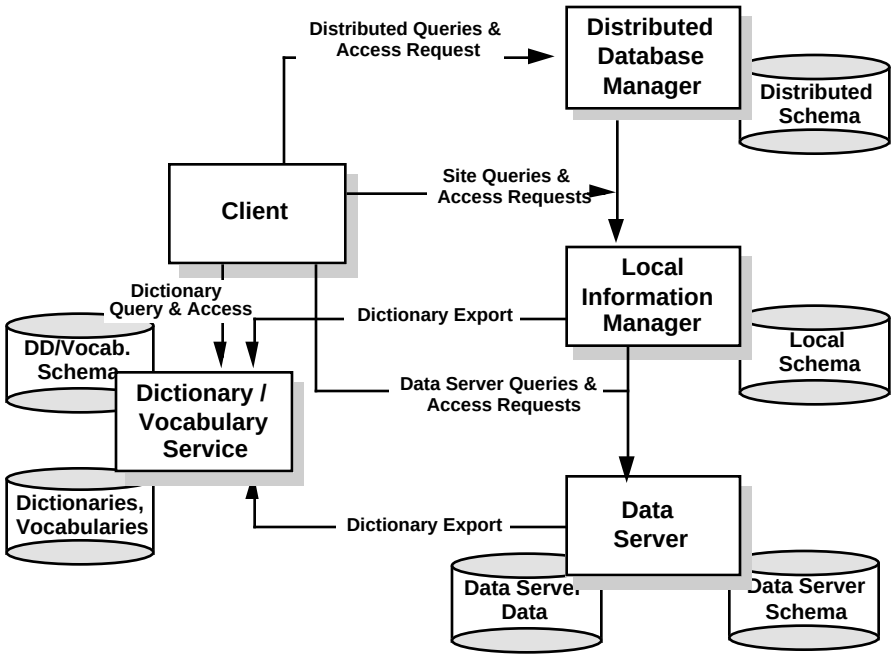

Figure 16. Database Management Reference Model

- *Local Information Manager (LIM):* an instance of a service which is capable of executing requests which require access to multiple data servers at a single site. LIM services can be requested by a DIM, or directly by a client. In either case, the LIM will act as an agent of its client: after the LIM accepts a site query or access request, it assumes responsibility for its execution and compilation of the result; the client can disconnect from the LIM, reconnect at a later point in order to determine the status of a query, obtain partial results, or cancel the query. The LIM uses a Local Schema to determine how a request should be executed. Note that a site may choose to provide several LIMs, perhaps supporting different data access and query languages. Different LIMs may provide access to the same data servers.
- *Data Server:* an instance of a service which is capable of executing data access, query, and manipulation requests against a collection of data[13]. Data servers can be accessed by

[13] Data server actually refers to a service provider at a logical level. The data server may actually be constructed from multiple physical servers implementing various aspects of the data server's functionality.

a LIM, or directly by a client[14]. Data servers use lower layers of data management services which are described in the data server architecture, section 6.4. Note that a site may choose to provide access to the same data via several different data servers, perhaps supporting different data access and query languages.

- *Data Dictionary Service:* a service which manages and provides access to databases containing information about data. Each data object, data element, data relationship, and access operation available via data servers, LIMs and DIMs are defined and described in the dictionary databases[15]. Data dictionaries are intended for access by users (e.g., to obtain a definition of a data item) and programs (e.g., to format a screen)
- *Vocabulary Service:* a service which manages and provides access to databases containing the definition of terminology, e.g., of words and phrases[16]. Vocabularies are intended for access by users (e.g., to obtain the appropriate term given a meaning) and programs (e.g., to let a user identify the intended meaning of a term which has several alternative definitions).

The model identifies the following key data objects supporting these services:

- *Schema:* a formal description of the content, structure, constraints, and access operations available for or relevant to a database or a collection of databases[17]. The reference model contains four types of schemas explained below. The elements which make up a schema will be defined through the continuing design effort. Schema information might be managed by the server using that schema, the data dictionary service, the advertising service, or all three.
- *Data Server Schema:* a formal description of the content, structure, constraints, and access operations for data accessible via a given data server. A data server schema is constructed from input provided by the data server itself. The same data may be defined in multiple data server schemata.
- *Local Schema:* a formal description of the content, structure, constraints, and access operations for data accessible via a given LIM. A local schema is constructed from input provided by the data servers which are accessible via the LIM using that schema. The same data server may contribute to multiple local schemata.
- *Distributed Schema:* a formal description of the content, structure, constraints, and access operations for data accessible via a given DIM. The distributed schema is constructed from input provided by the LIMs accessible via that distributed schema. The same LIM may contribute to multiple distributed schemata.

---

14 Architecturally, DIMs are permitted to access Data Servers directly, as well, but we currently foresee no need for this.

15 Note that the data dictionary database may be horizontally partitioned and the partitions may overlap. Each dictionary partition is called a data dictionary database, and will describe a subset of ECS data in the context of a particular science domain. Whether ECS will use a single or multiple data dictionary databases will depend on the result of the ECS data modeling effort.

16 ECS may provide several vocabularies, each defining the terminology of a particular science domain.

17 Note that this definition diverges from the ISO reference model by including the data access operations.

- *Data Dictionary / Vocabulary Schema:* a formal description of the content, structure, constraints, and access operations for data dictionary or vocabulary data.

The model provides for the following kinds of requests:

- *Query:* a query is formulated in a language offered (i.e., supported) by a DIM, LIM, or data server. It contains data search and access specifications expressed in terms defined either by the language itself, or in a schema offered by the corresponding server. In general, a query language is paired with a particular type of schema. For example, relational queries reference objects defined in a relational schema. The reference model contains three types of queries defined below.
- *Distributed Query:* a query intended for processing by a DIM. The query must be formulated in a query language which is offered by the DIM at one of its interfaces, and it must reference the corresponding distributed schema. A distributed query may, but need not, require access to multiple sites. The ECS DIMs may offer several query languages which still have to be defined. The DIM will process the distributed query and generate one or several site queries to satisfy the query. The DIM may further process the results returned by the LIMs to produce the final distributed query result.
- *Site Query:* a query intended for processing by a LIM. The query must be formulated in a query language which is offered by the LIM at one of its interfaces, and it must reference the corresponding local schema. A site query may, but need not, require access to multiple data servers. The ECS LIMs may offer several query languages which still have to be defined[18]. The LIM will process the site query and generate one or several data server queries to satisfy the site query. The LIM may process the results returned by the data servers to produce the final site query result.
- *Data Server Query:* a query intended for processing by a data server. The query must be formulated in a query language which is offered by the data server, and it must reference the corresponding data server schema[19].
- *Data Access Request:* a service request that invokes an operation offered by a DIM, LIM, or data server on a data object or a set of data. The object types and the operations which a given service offers for them are described in the schema used by the service. The reference model contains three types of access requests described below.
- *Distributed Access Request:* a data access request intended for processing by a DIM. The access request must conform to one of the interface protocols available from a DIM. The requested operation must have been defined in the schema used by the DIM for the objects referenced by the operation.
- *Site Access Request:* a data access request intended for processing by a LIM. The access request must conform to one of the interface protocols available from the LIM. The requested operation must have been defined in the schema used by the LIM for the objects referenced by the operation.

---

18 It seems likely that distributed and site queries will use the same query languages and the same protocols.

19 Data server queries might be subject to additional restrictions to simplify data server processing. For example, a data server might not support joins or might only allow joining of data objects of the same type.

- *Data Server Access Request:* a data access request intended for processing by a data server. The access request must conform to one of the interface protocols available from the data server. The requested operation must have been defined in the schema used by the data server for the objects referenced by the operation.

In addition, DIM, LIM, and Data Servers will conform to the general interface requirements as prescribed by the Interoperability Architecture. For example, this means that DIM, LIM, and Data Servers will accept requests regarding the status and estimated cost of a query or data access request. The servers also will use the Interoperability Services in the course of their interactions. For example, clients will use request brokers in order to locate a distributed information manger or a data dictionary service. This use of interoperability services will follow the general pattern defined in section 6.1, and is usually not explicitly mentioned in this section.

Data Dictionary Reference Model

A reference model for Data Dictionary Services and their relationship to schema maintenance is shown in Figure 17. The model is still under discussion and may change (the model for vocabulary services is analogous and not shown). It is a view of the data management architecture which emphasizes the components, interfaces, and data objects required by data dictionary services.

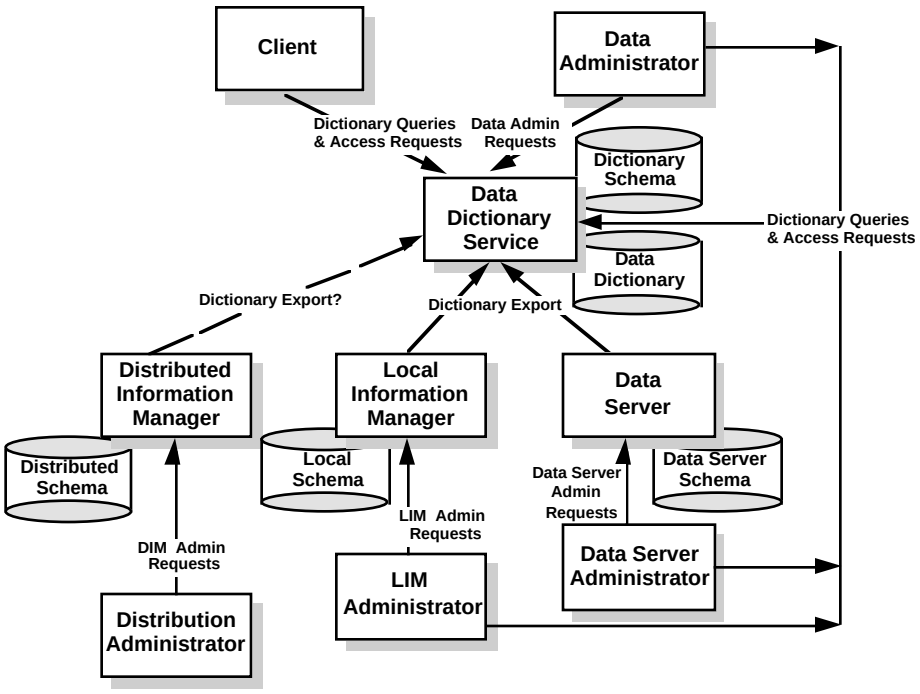

Figure 17. Data Dictionary Reference Model

The figure shows the following new types of services:

- *Client:* a program requesting data dictionary services. A client can be an application program or user interface which requires data dictionary information to formulate a query, or which displays explanations of data objects, data elements, or available operations to a user.
- *Data Administrator:* a service provided by a person or program to administer the data dictionary server, e.g., to maintain the data dictionary schema or to approve / reject proposed data dictionary information.
- *Data Server Administrator:* a service provided by a person or program to administer the data server, e.g., to define data objects and include them in the data server interface, or maintain the data server schema.
- *LIM Administrator:* a service provided by a person or program to administer the LIM, e.g., to define data offered by the LIM and maintain the LIM to data server mapping, or maintain the LIM schema.
- *Distribution Administrator:* a service provided by a person or program to administer the DIM, e.g., to define data distribution aspects and maintain the DIM schema. Administrator services may be separate software services (as depicted in the figure), in which case they are clients of the respective services with a special role. Alternatively, an administrator service may be an integral part of the service being administered.

The figure also introduces several new interfaces:

- *Dictionary Queries and Access Requests:* a query or access request intended for processing by the data dictionary services. Query requests must be formulated in a language supported by one of the data dictionary service interfaces, and must be compatible with the definitions in the corresponding data dictionary schema. The access request must conform to one of the interface protocols available from a data dictionary service. The requested operation must have been defined for the objects referenced by the operation in the data dictionary schema.
- *Data Dictionary Export:* requests for update of data dictionary information reflecting changes made to a data server, LIM, or DIM by their respective administrators.
- *Data Server Administration Requests:* requests for the creation or update of a data server, e.g., to define data objects and include them in the data server interface, or maintain the data server schema.
- *LIM Administration Requests:* requests for the creation or update of a LIM, e.g., to define data offered by the LIM and maintain the LIM to data server mapping, or maintain the LIM schema.
- *DIM Administration Requests:* requests for the creation or update of a DIM, e.g., to define data distribution aspects and maintain the DIM schema.

The following are some operational concepts underlying this model:

- Data Servers, LIM and DIM have administrators who are responsible for maintaining them. A data server administrator, for example, would be responsible for defining the

data types for which the server is responsible, adding new data types to the server, or altering the definitions of existing ones. Administrators are users assisted by software.

- Data servers export data dictionary information about themselves as new types of data objects or new servers are created or the definitions of existing ones are changed[20] in response to administrative actions.
- Local information managers export data dictionary information as part of the process (performed by the LIM administrator) which defines how they relate to data servers.
- The distribution administrator creates and administers a DIM. For example, it defines the LIMs which support this DIM. It is not clear whether a DIM would need to export any data dictionary information. This is because the sites (i.e., their LIMs and data servers) are the source of data definitions, not DIMs. However, it is conceivable that a future DIM might synthesize and then provide DIM-unique views of information distributed across several sites.
- The data dictionary service maintains the data dictionary based on the updates it received from data servers and local information managers.
- The administrators also maintain the schema of the respective server. Typically, server and server schema maintenance would interact with the existing data dictionary. At this time, our model supports several alternative scenarios:
  - The administrator interacts with the dictionary to maintain the object types managed by the server, perhaps using the definition of existing types of data elements and operations. The server updates the dictionary with information about the new object types. Concurrently, the administrator also updates the schema information managed by the server.
  - The administrator interacts with the server to maintain object type definitions as above. Upon completion of that process, that administrator (using utility software), updates the server schema combining the dictionary data describing the server with other information needed to complete the schema.
- The model does not mandate the separation of administrator and server software functions. The separation of the functions in the drawing merely indicates that the functions could be separated, and that they are different conceptually. In particular, administrator functions can be performed by the objects from which the server is constructed.
- Clients access the data dictionary via search and data access requests issued to the data dictionary service. Since the data dictionary is in essence a database, it is conceivable that the data dictionary service will present itself as a data server in the data management architecture, and support one or several of the data access and query languages.

---

[20] From an object oriented system architecture perspective, data servers are implemented in terms of the interfaces to the data objects they manage. Strictly speaking, the data dictionary information is, therefore, exported by these objects.

- The Data Administrator is a special type of clients whose role it is to administer the data dictionary, for example, by maintaining the data dictionary schema. Data dictionary information is assumed to be created through cooperation of data administrator and the administrators of the servers exporting data dictionary information.
- In accordance with the ISO model, the various types of schema and the data dictionary themselves are databases managed in accordance with the data management model. Logically, each server is a 'data server' for its schema[21], and the data dictionary service is a 'data server' for the data dictionary objects and their schema. Note that physical data management occurs below the level of the data server and is not shown in the model.

6.3.2 Implications and Issues

In the following discussion, we explore a number of implications and issues surrounding the Data Management Architecture.

An Earth Science Query Language

A key element of the data management solution is the selection of an Earth Science query language. Initially, the language will be based on a set of existing standard query languages supporting access to a complete range of Earth Science data[22]. The language will work in concert with a data schema. The data schema essentially defines the types of access and search operations which each data type and each site can make available. As the capabilities of the data servers and local information managers improve, the language will be able to support more complex forms of data searching including coincidence search and content based search, and such capabilities then can be included into the schema. Concurrently, the distributed information management component of the architecture will be improved to allow better optimization of requests among logically distributed data providers.

Operational Concepts: Intersite Search

In this section, we explore the ability of the data management architecture described above to support an array of intersite search requests. The current vision for intersite search capabilities is to support inexact coincident search across a bounded spatio-temporal region. The initial bounded restriction will evolve to unbounded space and time searches as processing and data access technology permits. The intersite search services evolve over time to assume greater responsibility for optimizing the complexities of intersite searches for coincidence.

---

21 The ISO model shows schemata and databases managed by 'database controllers'. The 'database server' logically corresponds to the 'database controller' of the ISO model. DIM and LIM are two levels of 'distribution controllers'; they will use 'database controllers' for the management of the distribution and local schema. However, we have chosen not to show databases, because physical data management occurs below the level of the data server; and we have chosen not to show the 'database controllers' for the distributed and local schemata so as not to divert attention from the fact that the responsibility for each schema rests with the respective DIM or LIM.

22 The CEOS Inventory Interoperability Experiment lessons learned document (January 1994) identifies some of the key characteristics that will be required in this protocol based on work performed in developing interoperability between V0 and european systems.

Complex searches such as, “What is the distribution of Phytoplankton Concentration underneath the ozone hole in the ice-free ocean for dates nearest October 15, 1995” will be captured in a data language that can express the relevant Earth Science data types and associated relational and type specific operations. Relationships across space and time will be expressible in an inexact but rigorous enough manner by the user to support parsing into data type value range searches by the search services.

Figure 18 presents an intersite search scenario representative of this type of complex query. In the example, the user constructs a query to determine the distribution of Phytoplankton concentration underneath the ozone hole in the ice-free ocean for dates nearest October 15, 1995. This query is decomposed into two subqueries and a results combination stage by the DIM. The first subquery uses TOMS data to generate an ozone hole mask for the desired dates. The second subquery performs a spatial intersection of the ozone hole mask from the first subquery with SeaWiFS data to generate a phytoplankton map over the ozone hole region. This map is then combined (perhaps at yet another computing site) with information about the extent of cloud and ice covers, to generate the final result in a date-tagged “product.”

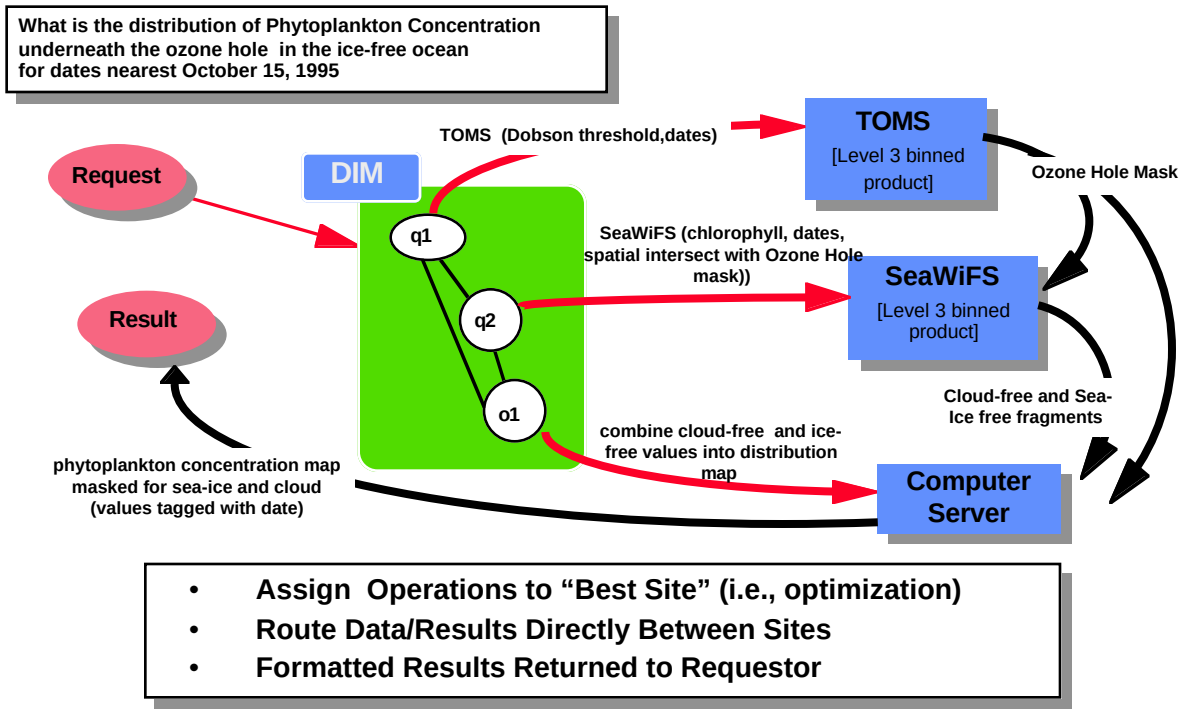

Figure 18. Intersite Search Scenario

General coincident searching refers to an inexact specification of joins of both indexed and non-indexed data referencing a combination of events, values, time, and space. To process such searches, the DIM must provide several distinct functions as illustrated in Figure 19. An *Intersite Search Parser* parses and verifies the request. The request is then analyzed by a *Search Planner* to determine how the query should be executed, i.e., it formulates a distributed access

plan. The plan is executed and managed by *Plan Execution* services. Site specific requests, which can include both queries and operations, are allocated to the respective sites for result generation. Each site takes responsibility for passing intermediate results to participating sites or back to the DIM services for integration and formatting of the final results. This is handled by the *Result Manager*.

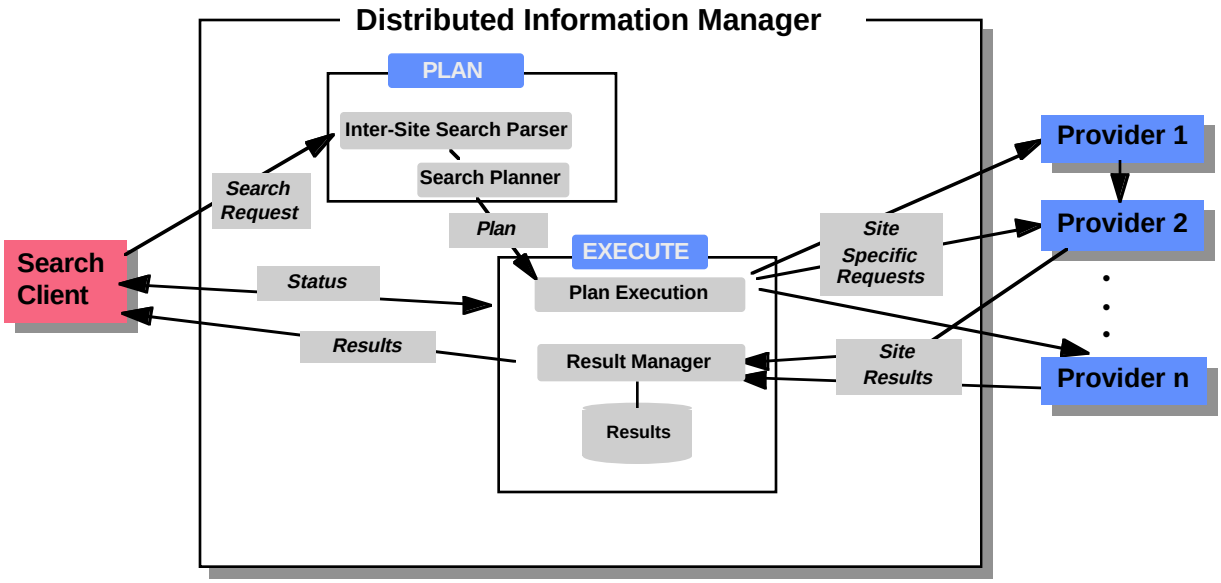

Figure 19. Intersite Complex Coincident Search

Figure 20 shows how interoperability services and data management services collaborate to accomplish intersite searching.

The user issues a request which has been marked as 'distributed search request' by the user interface. The request broker service, therefore, connects the user to a DIM service. Usually, there will be several DIM servers available and the request broker will pick the one most suitable, e.g., based on the proximity of the server, or perhaps because of the server's support for the search language used in the query. The DIM could execute at any site in the network, though it is likely that the user would be connected with one operating at the user's own site. Service providers advertise the DIM servers which they are making available, as well as their capabilities. In our example, the request broker would have used these advertisements to pick the DIM for the user.

Like DIMs, LIMs advertise their capabilities and their data holdings. The DIM, therefore, will also interact with the advertising service to determine which sites (i.e., which of the LIMs in the network) provide the data needed to answer the query[23]. If a single LIM can provide all the necessary data, the query can be forwarded to that LIM directly (i.e., the DIM can act as a search

[23] The architecture also contains provisions for a distributed schema which might be used on this occasion. The relationship between distributed schema and advertising service will be explored and definitized by the design.

agent even for non-distributed queries). However, frequently a query requires access to data managed by multiple LIMs. In that case, the DIM will decompose the query into its individual components (the subqueries) and issue them as new search requests. These requests can be routed to the various LIMs by request brokers.

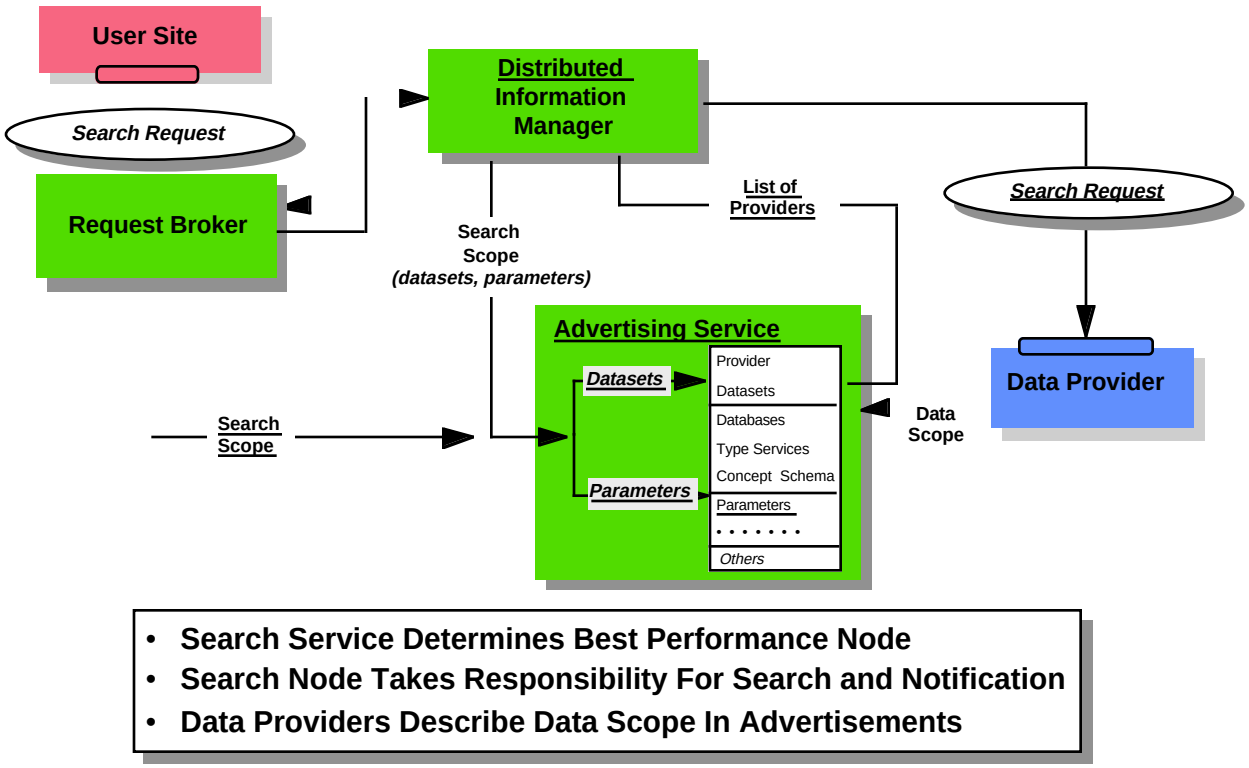

Figure 20. Intersite Search Routing

Issues

The query languages for DIM, LIM, and data servers for the various kinds of data still need to be selected, and the protocols for search and access operations still needs to be determined. There are conceivably four types of protocols for:

- queries of record oriented data
- queries of science data
- searches of document databases
- executing access operations (e.g., retrievals) by identifying the type of operation to be performed, the universal references of the objects to be accessed, and any additional arguments.

These protocols need to be flexible to allow different data formats/structures to be referenced, though some constraints on the range of data formats need to be specified to ensure that supporting the variety of formats is a practical proposition.

There is potentially a need for additional protocols to support updates (this is discussed more extensively below). The issues are related to

- the data architectures and what the OSI model calls the 'data modeling facility';
- the selection of COTS; and
- communications (if COTS is selected, it must fit into the communications architecture).

The data dictionary reference model and its role in the creation and maintenance of schema objects still needs more research. The responsibilities for distributing, managing, and providing access to schema information need to be assigned (i.e., which services provide access to schema data). An initial schema generation and update process has been proposed during design efforts; however, it is conceivable that the final design will not follow this proposal, e.g., if the relationships between data schema and data dictionary turn out to be more complex than anticipated. The interface between data dictionary and local and distributed information managers, data servers and data object types may not be determined until design. This may change the reference model presented here.

The data management architecture poses several issues related to data updating. First, it is likely that a "transaction" concept will be needed, but it is unclear which data management levels need to provide it. Data servers would use transactions to bracket sequences of operations they request of their underlying storage servers (where the storage servers support that concept). However, there may not be a need for DIM, LIM, and data server transactions.

The issue is related to the following. The SDPS architecture will need to support the creation and revision of data, but it is not clear which components of the architecture need to provide support to these operations. Updates will be received from the data ingest and production services, but it is less clear which other sources might process and send data updates. For example, users may wish to update objects directly, but the updates may not be sent directly to data servers because they require extensive processing to integrate (or re-integrate) the object into the overall database. That processing may not be part of the data server (e.g., it might be part of data production or ingest services). Such data would be 'loaded' or 'reloaded' rather than 'added' or 'updated'. For example, satellite data and documents will require re-indexing; earth science data updates or additions might need to trigger the calculation of other, dependent data.

It is not clear whether truly all data objects fall into this category (of needing 'loading'), or whether there are some data objects whose updates or additions could be processed completely by the data servers and would be bracketed as transactions. It is even less clear whether there are any requirements for (or any benefits in) passing updates through the DIM / LIM layer, as well. Research is required to clarify whether and under what circumstances this would be required or advantageous.

Key intersite search issues are the context mapping between provider sites and between the user and the provider sites, query decomposition, and optimizing the distributed access plan:

- The context mapping issue includes the problems of developing a common vocabulary or a translation between a multitude of vocabularies. Viable intersite search is dependent on a common mapping among vocabularies within a heterogeneous data management environment.
- The query decomposition schemes are difficult problems in distributed, heterogeneous environments, especially if the query language is extensible (as is the plan for ECS).
- The optimization of distributed queries is the subject of ongoing research and is not adequately addressed by current distributed database vendors. Separating the planning activity from the execution of the plan allows the near-term development of a non-optimized and perhaps manually generated plans. The resulting plan can be executed by the DIM services in the initial phases of ECS development. As distributed search optimization technologies mature, a more sophisticated planner can be inserted and tested for capability, reliability and performance without dramatically affecting the plan execution services.

6.4 Data Server Architecture

In the previous section, the Data Management Architecture was described. That architecture draws upon three key components: the Distributed Information Manager (DIM), the Logical Information Manager (LIM), and a set of data servers. The roles of the DIM and LIM components were described in the previous section. The architecture of the data servers are the topic of this section.

Data servers are defined as logically related collections of data and associated services. The data servers form a set of logical resources that can be managed and expanded on a site by site, discipline by discipline, and instrument by instrument basis. Data servers are ultimately responsible for the insertion, management, accessibility and distribution of products from the "product factory". Figure 21 depicts a conceptual model of a data server within the overall data management framework.

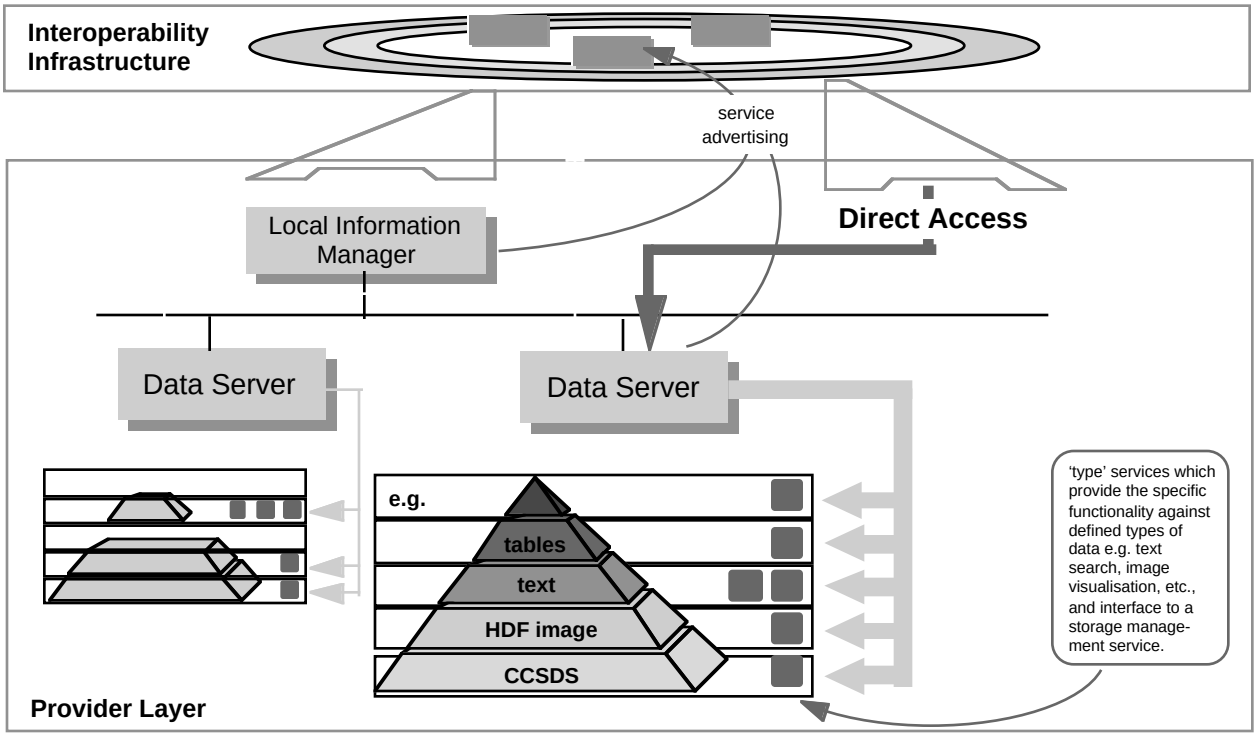

Figure 21. Use of the Data Server within the Data Management framework

At the heart of the Data Server concept are three key characteristics. Data Servers are:

- *customizable*, supporting site-by-site configurations;

- *extensible*, supporting the addition of new data types and services; and
- *evolvable*, allowing for the incorporation of new technologies as they become available

Data servers are customizable in that they are developed from a core set of data types and associated services defined for the Earth Science community. A particular data server can choose to implement any combination of those data types. There may be one or more data servers at a particular provider site (i.e., a DAAC or SCF). These different servers might vary in the types of data they provide, the range of services provided, implementation specifics[24], and even management policies.

For example, a data server might function purely as an ECS journal data server, providing abstracts and articles to the ECS community. This server might be part of a larger fabric of document servers that are distributed throughout a number of sites within EOSDIS. An SCF might establish a data server for specialized "research" products that local researchers would like to make available to a portion of the community. Because of the experimental nature of the products, contributors might want to tightly manage the availability of those products, ensuring, for example that appropriate information describing the products is delivered with the data itself. Downstream, value-added providers might establish fee-for-service data servers that ensure that appropriate security and accounting procedures are enforced during data access.

Data Servers are extensible in that through time their capabilities may be increased. The data types can be expanded within data servers by defining new data types and populating the relevant data server with appropriate services for that data type. This allows sites to augment the capabilities of a Data Server in response to users needs, without adversely impacting its existing user base.

Data Servers support the concept of evolution through encapsulating techniques and technologies within a layered model (See Section 6.4.1). As new technologies or algorithms arise in industry their incorporation is within a given component, minimizing the impact on the rest of the Data Server. As new and improved storage technologies become available, for example, the managers of a particular data server can choose to upgrade to that technology, without impacting other providers or its users.

As illustrated in the model, data servers may be accessed either through the Local Information Manager, or directly through an access method called "Direct Data Server Access". The Local Information Manager, as described in the Data Management Architecture, participates in single or multi-site search requests. These requests could return the data itself to the user; more likely, however, the results of a search would be returned to a user as a results set of items meeting the request criteria. A user would then employ Direct Data Server Access to obtain one or more of the data objects from that results set. This mechanism will be supported for all data objects in the system, from documents to sensor swaths, regardless of their type.

Through the use of Universal References, Direct Data Access is also employed in a number of other user services. For example, a scientist wishing to share interesting results need simply provide the appropriate universal reference to his colleagues, who may then obtain the data

[24] e.g., utilizing different system resources and supporting different storage technologies

through Direct Data Access. Additional services, including collaboration and subscription services, can also employ universal references and direct data access to provide functionality to users.

Access to the data server is via layered type services providing direct access at the lowest layer to either archival services for archived data, or to data production services for instantiated ("on demand") data. Figure 22 shows the role of these layered type services in accessing the underlying services of the "product factory", as introduced in section 4.2.2.2.

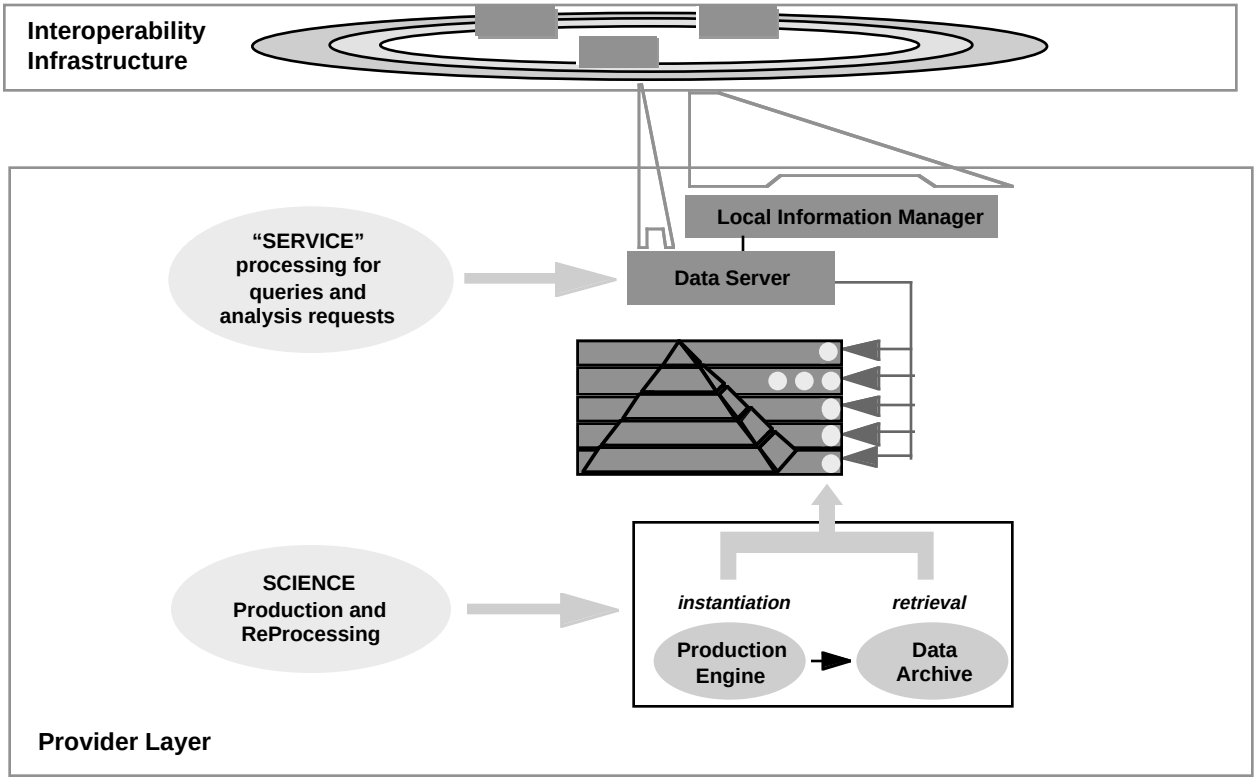

Figure 22. Data Access via Layered Type Services

Externally the data server is viewed as a layered model of the data collection with common services provided by the system and with user defined methods that operate on each layer. Data servers advertise the data types they manage and the associated list of data type services provided to the system via the distributed advertising services, described in Section 6.3. An important attribute of the data server interface is the polymorphic aspects of the common interfaces to each layer. These interfaces include operations that support the addition of data objects to the data server's collection, search, and a variety of data access operations.

The abstraction of production and archive services into a set of layered type services is an important one in preserving an evolutionary path to future data management advances. While

initially the data server might utilize a particular heirarchical storage management system for archival of some data types, new versions or different types of technologies may become available as these technologies mature. Thus we have developed an object-based abstraction in the form of data servers and data type services that will allow us to begin with current technologies now, and evolve to accommodate gradual evolution to future technologies.

6.4.1 Concepts and Reference Model

Figure 23 presents a layered model of the logical components of the Data Server. These layers are described in further detail in the following discussion.

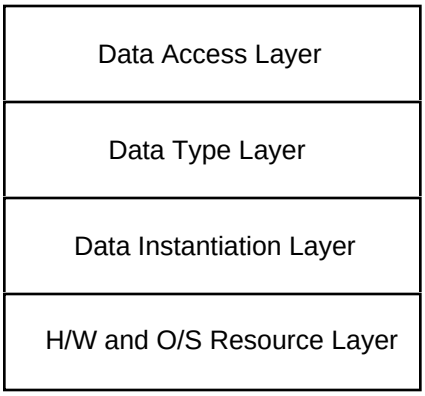

Figure 23. Data Server Layered Model

This layered model addresses the primary responsibility of the Data Server to store and provide access to data. In delivering data from the "product factory" to the data type layer, the Data Server may be able to extract that data from an archive, or it may be required to generate the data. Table 1 provides a classification of data objects managed by data servers that illustrates these two potential instantiation mechanisms.

	Archived	**Produced**
Advertised	Archived data products that are either ingested and stored, or computed and stored.	Products defined as "on-demand" by their providers. Such products are computed when requested, not explicitly stored.[25]
Unadvertised		Intermediate products generated in response to a user's request, for example as an intermediate result of a complex search operation.

[25] The use of on-demand products is an option that might be used based on the economics of compute vs. store tradeoffs.

In table 1, we note that unadvertised (i.e. temporary or intermediate) products are never archived. However, any product instantiated by a data server might be “cached” somewhere in the storage hierarchy. The caching strategy, including the use of distributed storage resources, is part of the data instantiation layer.[26] An expanded view of the data instantiation layer, showing the relationship between data caching, storage management, and data production as described in the data production architecture, is presented in Figure 24.

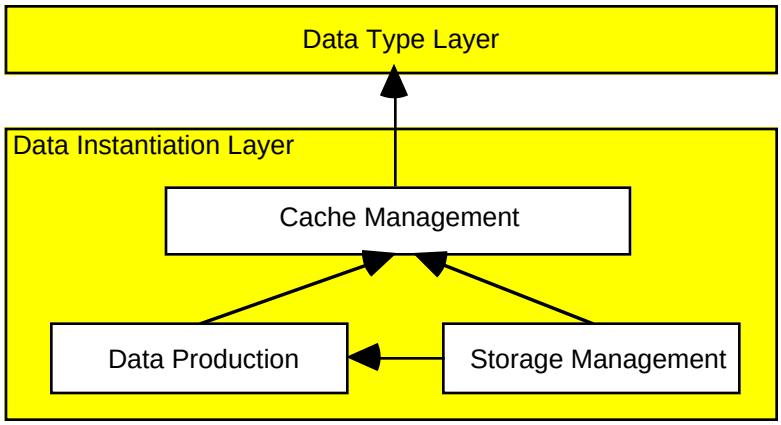

Figure 24. Data Server Layers Collaboration with ECS Data Production

Each of the service layers within the layered Data Server model utilizes the services of the next lower level. An important feature of this layered architecture is the concept that the services provided by each layer may be polymorphic. This means that the service is provided to its requester regardless of the fact that the service may be fulfilled in different ways for different data types. This is where it becomes important to distinguish between the *services* provided to users at any level and the *technique* by which these services are fulfilled. While it is critical for users to be aware of the services that are available to them and to semantics of those services, the actual technique of meeting that service is abstracted away. For example, inspecting a document and inspecting an image will be performed in very different ways. The user only needs to know that they can inspect these different types of data.

The services that the Data Servers provide can be classified in one of two types: data oriented or system administrative. Examples of those that are data oriented include services to add new data to the data server’s collection, retrieve data, search data, update data, inspect data and to execute

[26] Note that a data server might even employ a user's storage resources in "caching" data products. This allows, e.g., at an SCF, for the staging of data products close to the point where they will be accessed. We note also that caching might be provided either explictly, e.g., by creating a copy of a data object on magnetic storage, or implicitly by the underlying data representation technology, e.g., an object oriented database system.

a computational method upon one or more data objects[27]. These services are met by the Data Access layer and are described in more detail below. Examples of system administrative services include those to startup, shutdown and restart the system, manage user sessions (login/logout) and configuration. These services will be developed in more detail in the Data Server logical modeling effort.

Data Access Layer

This layer presents the services that are fundamental to the role of the Data Server in ECS. The services of this layer are those utilized by all users or clients of the Data Server. These services are the vehicle providing access to all of a data server's advertised data products, whether computed on-demand or retrieved from archive. This layer provides a consistent set of services for all data types. A responsibility of the Data Server layer is to ensure that as data is ingested and placed in the Data Server's collection, its availability and all the services that it can provide are advertised to the Data Management/Access component of the ECS.

As a result of user and data modeling efforts, a complete set of services that are useful for all data types will be defined. Currently the services identified in the following paragraphs represent those that the Data Server layer will provide that support the science data user directly. Other data oriented services may be identified as a byproduct of the data modeling effort. There will be other services that will support the administration of the Data Server.

Insert service

The insert service will allow new data objects to be created within the Data Server. The new data objects will be created from data that is supplied either electronically or on other media (e.g., CD-ROM, 9-track tape, 8 mm tape cartridges). The data supplied to the Data Server must be of a type known to the Data Server. From this known type the Data Server layer will utilize the services from the Data Type layer to archive the data object.

Retrieve service

The retrieve service will allow users to obtain data objects from the Data Server. Users will only need to ask the Data Server for the given data, specifying the form in which they would prefer the data to be delivered[28]. Typically the retrieve service will be utilized by other ECS components, but this service will be available for the support of direct data access by users, bypassing the Data Access/Management ECS facilities. This will be available to users when the user knows what data object they want, and in which data server it resides. This is analogous to a user of a file system knowing exactly which file is desired and the path to that file. The user then does not need to perform a directory search, but rather can access the file directly.

Search service

The Search service will allow users to search for a given data object by some appropriate criteria. This applies to all data types, even though the mechanism for searching the various data types

---

27 Data servers provide this execution capability on behalf of the data production facilities, yielding a common means of accessing both stored and computed data.

28 e.g., electronic transmission, CD-ROM, tape form factor, perhaps specifying one of a number of data formats as well.

and the search criteria specified is type dependent. This service will utilize the lower level services within the Data Type layer.

Update service

The update service is similar to the insert service, allowing an existing object within the data server to be updated. The value of this service will be in allowing "versioning" of data objects, making the updated version of the data object the "current" one while still allowing access to previous versions. The mechanism by which a particular data server chooses to implement "update" capabilities is left to the implementation of the specific data server. A data server might choose to maintain a complete copy of the modified object (i.e., even the unmodified parts), or might only maintain "differences", complete with an audit trail that supports generation of any version from these differences.

Inspect service

The inspect service accesses a special attribute of data objects that supports a data "browse" object[29] or an inspection of the complete data object. Examples of what this service will supply include a browse access facility for some data types and a scanning capability for others. The effects of utilizing this inspect service will be data type dependent. For some data types there might be multiple inspect services with different capabilities; for example one service might show a graphical summary of a data object while another might allow the user to view the actual data values. For some data types this service will involve an interactive session between the user/client and the data server resources. Again, this Data Server layer service will collaborate with lower level services within the Data Type layer.

Execute service

The Execute service will allow users to request that a given science algorithm be executed upon one or more data objects. The science algorithm to be applied might be one within the data server/data production facility or it might be a user provided algorithm. A number of the services that are available to users on all data types will be available on the standard Data Server services, examples of which have been addressed above. However, there are other services that are available on some data types. These additional services may be either built-in services specific to the data type that are extensions to the standard services, or standard science data production algorithms. The Data Server invokes data production services on behalf of the user in fulfilling these requests.

Data Type Layer

This layer represents the services that each data type can provide. Many of the higher level services of the Data Access layer will be directly supported by the same service at the Data Type layer. This is where the Data Server will take advantage of the benefits of polymorphic interfaces: The Data Access layer services are the same for all data types, but the Data Type

---

[29] This concept is analogous to that of data "abstracts" in the Sequoia 2000 Project. It is applied universally across data types, not just image types. The concept supports the notion of an "at a glance" feature that might be used to give a user an indication of the content of a data object without having to view the entire contents of that object.

layer services will perform those services differently on a case-by-case basis. As an example, the Data Access layer provides a search service for all data types. In the Data Type layer the difference between searching for specific text in a set of documents and searching a level 3 data product for rainfall over a certain rate is accounted for. The search criteria will be embodied within the search request. Those criteria may be very different between different types of search requests.

The complete set of services provided in this layer are being determined within the ECS user and data modeling efforts. These efforts will specify exactly what the user community needs from each data set, and how that need should be addressed. At a minimum, the data oriented services that are available at the Data Access layer will be provided for each data type in this layer. Examples of additional Data Type services include the services to compress/decompress and encrypt/decrypt the data objects. These services may employ data production services in accomplishing their task.

Data Instantiation Layer

The data instantiation layer is responsible for delivering the specified data products to the data type layer. In doing so, it may access one of the data server's persistent storage archives, or may arrange for spontaneous generation through the production services. It can employ portions of the storage hierarchy to cache data objects, providing more efficient access to frequently referenced data objects.

This layer represents those services that any given persistence technology must support. These services are those needed to save and retrieve data. This is another opportunity to take advantage of polymorphic services. In this case, the services of the data instantiation layer will be met differently by each persistence technology implementation.

This layer represents the translation of each data type's storage and retrieval services into the proper persistence technology implementation's services. This allows variety in how data types are stored. This variance has two dimensions. The first is that different data types may be stored by different persistence technologies. For example, documents may be stored in a hypertext data base, whereas large level 0 granules may be stored in a hierarchical storage management system. The second dimension of the flexibility afforded in this layer is that the persistence technology employed for a given data type in one data server might be different than that utilized for the same data type in another data server. Additionally, this flexibility supports the extendibility of persistence technologies with minimal to no impact on the existing production system.

The persistence technologies that are available to Data Servers currently include relational databases, hypertext databases, hierarchical storage management systems, object databases, simple files and the possible standards that technologies like Andrew File System (AFS) and OSF's Distributed File System (DFS) bring. In providing the Persistence layer with a consistent set of services, the Data Server can evolve as new technologies that support the persistence needs of ECS emerge. A simple interface that maps the data instantiation layer services to the technology will allow that technology to be inserted into a Data Server. In this area we may have the opportunity to take advantage of developments in COTS object technology, for example in

the basic object services of the OMG Services Architecture, including Consistency Control, Backup/Restore, and Replication.

Another important feature is that a Data Server will be able to support multiple persistence storage mechanisms concurrently. For example, a Data Server might have a hyper-text database for support of all text data types, an object oriented database management system (OODB) to support browse images, a UniTree system for some Level 0 data types and a separate EMASS system for Level 2 data types.

Hardware and Operating System Resource Layer

The operating system and hardware resources will be used to support all the Data Server services. It is expected that many of the services that are needed out of this layer will be provided through COTS, collaboration with the System Management component and the native Operating Systems themselves.

6.4.2 Implications and Issues

In the following discussion, we explore a number of implications and issues surrounding the Interoperability Services Architecture.

The current level of attention to the "direct data access" need should be broadened. This issue must be evaluated to ensure that it addresses the purpose of giving users "direct data access." Currently this equates to giving users the ability to go directly to the correct Data Server, and its services, to access data. Any lower of a level data access, such as directly into the persistence technology introduces a large number of issues, many having to do with the management of storage and processing resources. The implications of providing lower level access to such services require additional investigation.

At this point of the architecture's development it seems clear that there are at least several candidates for reuse within ECS. From the Data Server's perspective two candidates are a software component that supports resource management and another that performs scheduling. There might easily be reasonable needs for the Data Server to perform these functions in support of its own internal responsibilities. If the Data Server could utilize the software components that will be constructed for other portions of ECS, the Data Server implementation would be simplified. However, this will require some additional coordination during design and development to ensure that a common facility can be delivered that meets the needs of both production services and the Data Server.

The timing of COTS software and hardware will continue to be an issue for Data Server implementation. The software COTS issue includes various database technologies, as well as hierarchical storage management (HSM) systems. We anticipate continual advances in these areas over the life of the project, and will need to give considerable attention to supporting technology evolution in these areas.

As described above, the Data Server will collaborate with the data production services as presented in the Data Production Architecture, section 6.5. The manner in which the data server interacts with the set of production services will be resolved through the continuing system

design effort. Specific issues in this domain include extending the system by incorporating new services. How these new services are known both to the Data Server and to the Data Management facility must be determined. Another specific issue pertaining to this relationship has to do with the manner in which shared resources are managed. A corollary to this issue is how the users might make some of their resources available to the Data Server/Data Production facility, for example to supply caching resources "close" to the point of eventual use (e.g., at an SCF).

Another facet of the extensibility of a Data Server is the incorporation of new data types. Exactly how these are specified and brought into the set of data types known by the Data Server must be established. The strategy for identifying the viable production services relevant to this new data type and how the data type is "linked" to the production service must be identified. Finally, given the new data type and the services that it provides, there is the issue of how the data and services will be advertised users and to the data management components (DIMs and LIMs).

Data Server support for version control of data is also an issue. Different persistence technologies may support this in different ways. Techniques may also vary based on data type. Which data types or technologies support or require full replication of data objects and which allow for versioning through incremental change maintenance is an issue to be addressed in more detail during design.

Finally, there is the issue of archive envelopment. In archive envelopment, a Data Server would be extended to include an existing archive, (that is the storage resources, management software and archived data), so that the existing archive becomes part of the Data Server[30]. In this way the user could utilize the same Data Server services to access the data in the pre-existing archive. The tradeoffs of providing such access via archive envelopement vs. via explicit heritage system interoperability still need to be enumerated and analyzed.

[30] In particular, this concept has been discussed as a means of incorporating Version 0 archives into ECS.

6.5 Data Production Architecture

The data production architecture concerns the ingest and processing of data. The ingest function covers the reception of all types of data into ECS and the loading of the data and data/information derived from it into data collections which can be accessed through the data server.[31] The processing concerns both the routine and on-demand processing of science data and ad-hoc processing required to service user requests.

There are several classes of production that need to be considered within the ECS architecture, as shown in table 2 below.

	Planned (routine)	**On-demand**
Standard Product Processing / Reprocessing	EOSDIS approved products with high anticipated use or required as input to other standard products (DAACs only)	EOSDIS approved products with low anticipated use (DAACs only)
Special Processing	Routine EOSDIS research product generation at SCFs and DAACs (may be for limited period)	Ad hoc research requests to SCFs and DAACs
Request Processing	Request processing which is too large to be dealt with on-demand. Scheduled to make optimum use of resources in conjunction with other planned activities	Processing for ad-hoc user requests (content search, subsetting, reformatting etc.)

Table 2. ECS Production Summary

The distinction between these classes arises from the response required from the processing, as well as aspects of predictability in the processing: science processing is schedule driven, from standing orders and production requests; request processing is essentially on-demand. The architectural concepts required to support the classes of processing are, however, similar in many respects.

Local policy will dictate the split of data products to be produced as planned 'standing orders' and on-demand requests. This policy will be based on expected usage of a product and resource cost analysis for particular data collections. The data production architecture must support the variation in policy from service provider to service provider and from data collection to data collection, and is changeable over time (e.g., as usage patterns change).

[31] The data server is discussed at length in the previous section. The set of data objects under the control of a single data server are referred to here as data collections.

6.5.1 Concepts and Reference Model

The overall organization of the production architecture into three component services is shown in Figure 25. These services will be considered as separate sub-architectures in the later stages of analysis and design. The three services are described at a high level below:

- The *data ingest service* is responsible for receiving all types of data into ECS which needs to be archived and made accessible to the user community. It will plan whatever processing and loading services are required to archive the data and make it available with associated metadata. It will then make requests to the data processing service for scheduling its plan onto the processing capacity, and to the data server to access the services for loading data into the archive.
- The *data processing service* is responsible for scheduling the requests for processing onto the configured processing resources. It uses the data server service to request staging of data onto the relevant storage area. It sends acknowledgments to the requesting service when the processing is scheduled and when it is complete. It also provides status messages, schedule views and references to the output from the process.
- The *data server* provides access to services which act directly on a data collection to support the data ingest and data processing service. For example, it would provide access to database management systems used to hold data and metadata for the update process. It also provides the access point for a user wishing to request the status of a particular process within the schedule or during processing.

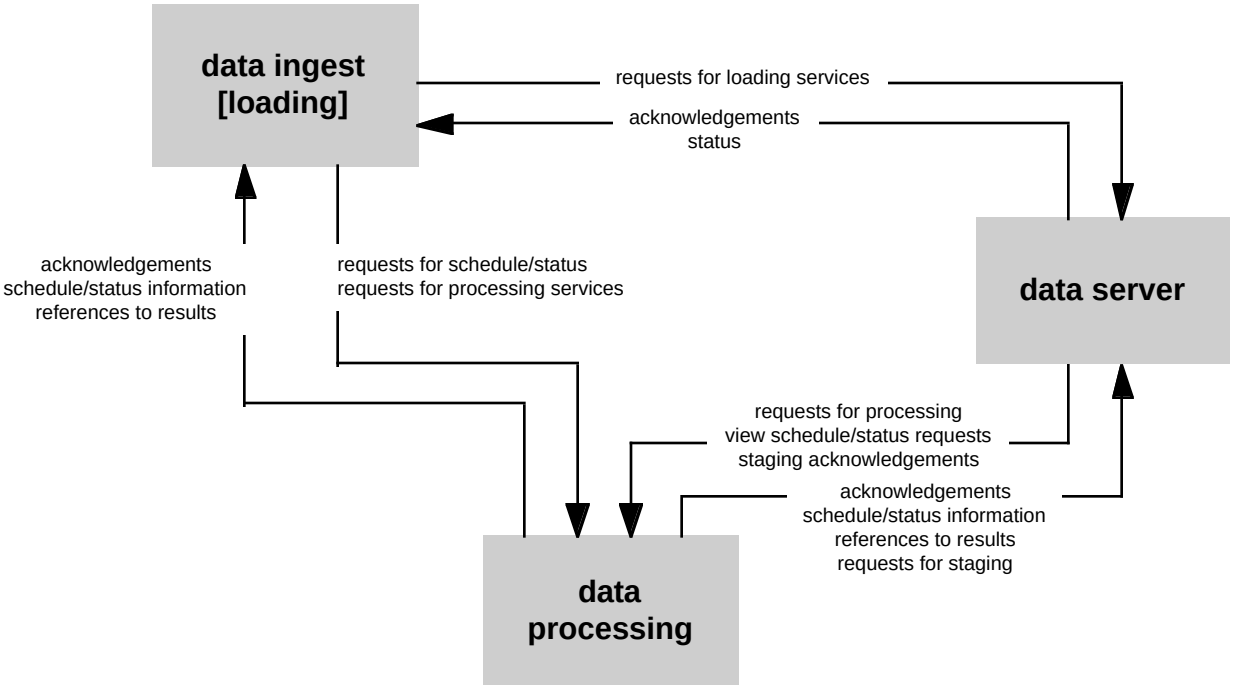

Figure 25. Interaction between services within the Data Production Architecture

Data Ingestion Service

There are various types of data to be ingested into the ECS. These can be divided into the following classes:

- L0 Science Data

  Data received from EDOS from the EOS spacecraft, and Pacor from the TRMM platform. This group includes the delivery of associated platform ancillary data.

- Processed Data

  Data (products) from other DAACs and SCFs used in the production of other products or satisfying service requests.

- Auxiliary Data

  Data from sources other than the EOS spacecraft used in the processing of science products (e.g., digital elevation models, meteorological forecast data). Several classes of auxiliary data have been identified[32]. In general the data is provided in spatial and time frames which do not match the data from the EOS spacecraft and therefore needs pre-processing to be used efficiently within the science processing.

- Documents

  ECS will receive documents related to specific objects within ECS (e.g., algorithms, products, etc.), research undertaken using ECS , and of general Global Change interest. These need to be archived and indexed within ECS and where appropriate associated with the relevant ECS data collection.

- Correlative Data

  Data used for calibration, validation and verification of EOSDIS science products. Similar in character to auxiliary data, these datasets will generally be archived in their "as-received" format with appropriate metadata to assist in location of suitable data for specific analyses.

- Algorithms/User Methods

  Methods which form the executable processes used by the processing service. The ingestion of these objects is discussed below.

Although the details of handling each type of data during ingestwill vary between each class, the general process will be the same. That process can be described as follows:

- On reception of the data the data ingest service will be used to load the data into a 'working' data collection; essentially a temporary archiving step, to place it under some

[32] As discussed in: Ancillary and Auxiliary Data in the EOSDIS Core System (ECS) Issue 1.0 September 1993. ECS Document ref. 193-00123.

control and inventory. This process may be automatic on receipt of data into an ingest buffer or initiated manually for data not routinely ingested.[33]

Prior to the loading some data specific quality assessment function will be required to determine that the data received is what it purports to be in the delivered metadata. A simple data server will provide access to the services required by the data processing service and data ingest service. These services must support inventory querying and management and transfer of data from device to device (for staging) as a minimum.

- Having registered the availability of some input data, the data ingest service will construct requests for its subsequent processing and archiving based on the rules generated by the DAAC management. This process will include consulting data dependency rules to establish whether all required input data is also available. This will result in the definition of a number of parallel or serial processes. Dependencies between separate processes must be captured in the definition of each process, so that they can be scheduled appropriately by the data processing service.

  Data production plans are generated in advance of the data arrival using data availability schedules. These plans are used to reserve processing resources in order to guarantee a given level of service for the production processing. [34]

  It is possible that some processes will not be known at the planning stage. For example, an algorithm might need to initiate a separate process depending on the some intermediate result. The planning of this type of unpredictable set of processes needs further analysis.

- Having established that all required data is available and which separate processes need to be performed, the data ingest service passes the individual data production requests to the processing service, which schedules the resources and delivers an acknowledgment that each request is scheduled. Each production request must include information on dependencies with other requests.

- The data processing service allows the DAAC operations staff to view the processing schedule and make adjustments to the schedule by changing priorities and requesting a reschedule. The data processing service also supports status requests during execution.

- Once the acknowledgment of successful process completion is received from the data processing service, the data ingestion service requests loading services from the data server to place the data and its associated metadata into the relevant data collection(s) which are accessible by ECS users. This process will include:

  - transferring the data from the processing staging area to the archive media/device

---

[33] There are still several inconsistencies in the details of the EDOS - ECS interface. This is being resolved through joint IRD/ICD development by the two teams.

[34] The production planning process needs to be configurable by the DAAC staff. For example in some situations ancillary/auxiliary data will be pre-processed as part of the same production request as processing the associated science data, in others it will be pre-processed separately as soon as it is received.

- transferring the data (or its reference) to the data dissemination service for onward transfer to users who have supplied subscriptions to the data in question, or orders for the data.
- reformatting metadata generated by the process into a loadable form
- performing data management quality checks (general and DAAC specific)
- loading new metadata into the collection
- modifying metadata already in the collection[35]
- logging the addition of data

This process applies to a provider using the 'process then store' approach to support data supply. For providers wishing to support the 'process on-demand' approach then only the lower levels of data are stored and the metadata will be updated to indicate virtual products, i.e., products that can be requested but haven't been processed yet.

Data Processing Service

The data processing service receives requests from the data ingest service and the data server for processing. Before the requests are issued these two entities will verify that the data required by the request is available. The requests from the data server will include both on-demand science processing requests and processing required as a result of accessing the data collection through the data server (e.g., a subsetting task).

The inter-relationships between the services comprising the data processing service are shown in Figure 26.

[35] For example, the directory metadata might include the end-date for the available data set; this will need updating as new data is added.

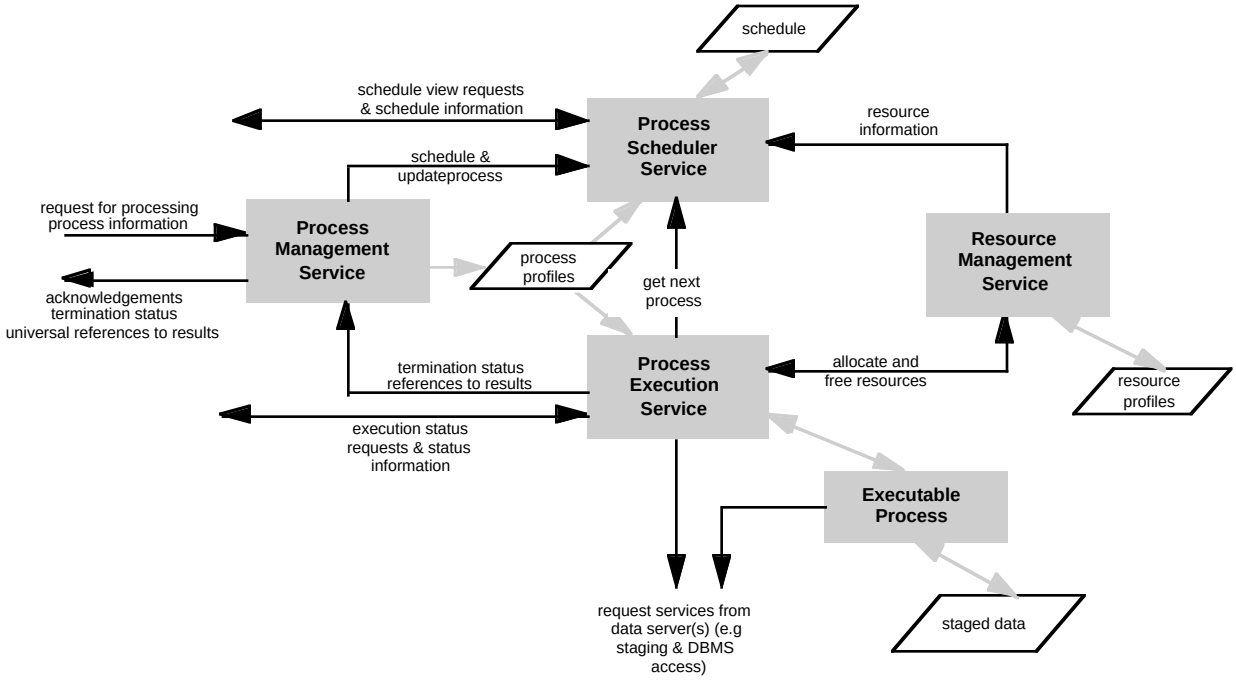

Figure 26. Conceptual architecture for the Data Processing Service

On receipt of a valid processing request the data processing service schedules the processing based on information it receives as part of the request (processing profiles[36]); knowledge of the processing resources available (resource profiles) and rules for prioritizing requests. The service provider can adjust the resource profiles and prioritization rules dynamically and ask for a rescheduling of processes in the current queue. The scheduling will take account of the process dependencies implied in the request (i.e., process A must complete before processes B and C can start).

The working schedule for the processing request can be viewed by users through the data ingest service and the data server. Authorized users can cancel a current process or one that has been scheduled if they were the originator of the request. The data ingest service also supports re-prioritization and rescheduling of processing requests which are in the schedule but which are not completed.

The overall schedule for the processing service may be viewed by authorized users (such as facility managers). These users may have the authority to cancel requests even if they are not the originators.

---

36 For some processes these will be well known (e.g., routine science processing), for others, such as request processes incorporating user methods the profiles in terms of the resources required will be inexact. Dealing with this variability of information is an important issue for the service.

Once a task is scheduled the resources are allocated and at the appropriate time the relevant input data files and executable process are staged onto the processing resource. The staging is requested through the data server dealing with the 'working' collection for the data files. The staging process might involve the physical transfer of the data to the staging area, or might simply connect to a database for data that is used by many processes[37].

During execution the data processing service can provide status information to requesters. This might include progress, summary results, or error conditions

Quality Assessment of data may be performed during or after processing. During a process it forms part of the executable process and needs to be included in the algorithm or method. After a process, it would be performed as a second 'dependent' process following the first process. There are two methods for quality assessment: routine or automatic QA, using a predetermined algorithm; and QA requiring scientist or operator analysis of the data (e.g., product visualization)[38]. The subscription service concept described above will be used to notify algorithm developers when certain characteristics are identified in the QA process, which have previously been defined as of interest to the algorithm developers.

When a process is complete the execution service sends a termination status message to the process management service, with references to the input and output products. The process management service passes this information to the requesting service, but converts local references to ECS universal references. The process management service also requests the scheduling service to update the schedule with the completed task[39]

The data processing service needs to be developed independently of the following:

- the hardware architecture and classes of hardware used in the processing string(s), so that each provider site can scale in a modular fashion
- resource specificity of a process; the service must be able to establish which resources a process must have to complete successfully. This will support sites which allow executable processes to be moved from one platform to another and other sites which dedicate some resources to a particular routine process, for example an algorithm requires a particular type of hardware (e.g., MPP) or it is more efficient not to stage an algorithm and associated static data files on and off a resource between initiations of a process.

Algorithm/Method Ingestion

There are essentially two classes of processes used in the data processing service; science processing algorithms and methods. The ingestion of these two types into the ECS are described below.

Science processing algorithms are all developed externally to the ECS, usually within Science Computing Facilities. Developers within the research community provide algorithms to the ECS

---

37 This latter approach may be useful for example in providing access to DEMs or orbit ephemeris through DBMS mechanisms. The interface from the process would be implemented through the PGS toolkit (see Section 5.4.4)

38 The impact of human intervention in the QA process during processing is an issue that requires further study.

39 The issue of whether the schedule should be updated through a request to the scheduling service from the execution service of the process management service requires further study.

to process L0 science and processed data ingested at a provider site into standard and special products. The provider site manages the resources and operational management required to support this processing.

To the algorithm developers, the Production Architecture represents a production environment in which the algorithms are executed. Toolkit routines are made available to the algorithm developers which encapsulate the interfaces between the algorithm and the production environment. This is the mechanism by which the algorithms interface in a consistent manner with the production environment. The specific elements of the toolkit used at each provider site will vary.

To support the ingestion of science processing algorithms, an Algorithm Integration and Test Environment (AITE) is provided at DAACs to make available staff and services to the algorithm developers: to ensure algorithms meet the safety requirements of the site; and to integrate the algorithms into the Production Architecture. This facility is responsible for establishing the algorithm within the Production Architecture, including the configuration management, advertising (locally and/or globally) and registration of algorithms within the processing environment.

The second class of processes, methods, provide fundamental services to the data server architecture described in Section 6.4. Three categories of methods are recognized: standard ECS methods; reviewed methods, and user methods.

- *Standard ECS Methods* include the data methods, such as subsetting and reformatting, developed by ECS to facilitate data access requests through the ECS services.
- *Reviewed Methods* are methods delivered from outside the provider site which need to be integrated as one of the services offered to the whole user community. For example this type of method may perform specialized data searches on one type of data. Such methods may be delivered through some sort of peer review process. They would be integrated into ECS through an integration and test process similar to the science processing algorithms described above to ensure safety and reliability levels are met. The integrated user development methods would then be advertised as part of the provider services.
- *User Provided Methods* are algorithms which are delivered to the data server by a (non-ECS developed) user as a part of a service request. The Data Production architecture should have the flexibility to accept this type of method, although there are many issues about the safety and reliability aspects of supporting user methods. Further analysis will be required to establish at what level user methods can be supported within the ECS.

6.5.2 Implications and Issues

In the following discussion, we explore a number of implications and issues surrounding the Data Production Architecture.

The major issues to be resolved for the data processing architecture are mainly related to the implementation architecture and therefore are not considered in detail in this document; i.e

- the characteristics of the ECS interface to EDOS and PACOR
- hardware configuration for supporting the processing

These issues are the main subject of the current ECS performance modelling (both static and semi-dynamic 'strings' modelling) leading to the generation of an implementation architecture. While the processing requirement is so variable it is important that the architecture retains its flexibility to deal with capacity changes, both up and down. Massively Parallel Processing architectures offer a number of potential performance advantages, but may compromise architectural flexibility to meet future performance requirements. The trades between performance offered and flexibility lost need to be performed; in conjunction with the science and DAAC communities.

The need for a scheduling service and planning/management services for the data processing has been identified. These need to take into account the full range of variability expected from algorithm operations concepts. Unfortunately these concepts are not yet available uniformly across the algorithm development community and therefore it will be difficult to develop a design for these services. ECS will attempt to verify its design for these components with the emerging plans from the instrument teams. The project will also look at commonalities between scheduling/planning services elsewhere in the architecture (e.g DIM, FOS, etc.)

One of the most challenging issues associated with the data production architecture is that of resource management, as data production resources are shared among a variety of potentially competing tasks. Production resources support the generation of standard products, both according to production schedules (in response to data ingest), and in response to user requests (for on-demand products). Additionally, production resources are available for the development of special or research products under the direction of a suitably authorized user. Finally, production resources may be employed in generating implicit products or in providing computational support for queries (e.g., in content-based search). The manner in which a site's production resources are managed in this environment of conflicting requests requires additional attention.

Additionally, it is important to allow a user's (e.g., an SCF) computational resources to be applied to the production of certain products. This concept serves to extend the resource base of ECS, providing computational support "close" to the point where data will be used, and taking advantage of significant computation resources that exist within the scientific community. Effective use of an SCF's computing (and storage) resources in a string of analysis requests might significantly reduce the burden on ECS system components, including archive robotics and drives, and computational resources. Additionally, effective computing strategies may drastically reduce the amount of network traffic required to satisfy a user's request. For example, a researcher exploring the effect of various spectral mixing methods on a particular dataset might repeatedly operate on the same level 1 data products. Rather than accessing the level 1 data products from the archive each time they are needed, then computing custom level 2 and 3 products and shipping those across the network, it might be preferable to use the resources

at the researcher's facility to provide a more effective analysis engine. The level 1 data could be "cached" at the researcher's facility, and local computational resources could be used to provide a streamlined analysis process.

This concept could also be used to bring large amounts of computing power to bear on "high profile" problems on an as-needed basis. For example, the nation's supercomputing resources (at NSF supercomputer centers and other high performance computing centers) might be employed to run a series of models in response to a phenomenological event.[40] Issues related to the control and management of "contributed" resources require further investigation.

Finally, the issue of dealing with user methods dynamically delivered within a service request, in a provider site needs further investigation. Prediction of resource requirement, priority allocation, non-interference with routine production, error/exception handling and security are all issues that need to be addressed before adequate architectural provision can be made for this functionality and indeed the relevant requirements can be formulated for the System Requirements Document.

[40] Such capabilities might have been useful towards the tail end of the Desert Storm operation, when Iraq forces set afire hundreds of Kuwait oil fields. At the time, there was great speculation in the effects such an environmental catastrophe might have. The ability to harness a large amount of computational power for modeling efforts in such environmental crises might be supported by appropriate resource management provisions.

6.6 Communications and Interconnection Architecture

The preceding sections introduced a number of SDPS sub-architectures in the form of a client-server model, i.e., as a collection of services. This section discusses how the augmented client-server model described in section 6.1 fits into the overall ECS communications architecture as supported by CSMS.

Figure 27 presents a stacked model of the earth science user-provider model presented earlier. Services either support end user operations (e.g., user interface services), provider operation (e.g., data management services), or interoperation between users and providers (e.g., distributed information management services). When this view is overlaid with a client-server[41] model, it is obvious that 'clients' and 'servers' exist in all three of these layers, and that the 'client interfaces' and 'server interfaces' of the SDPS applications view are both essentially client-server interfaces according to the client-server model.

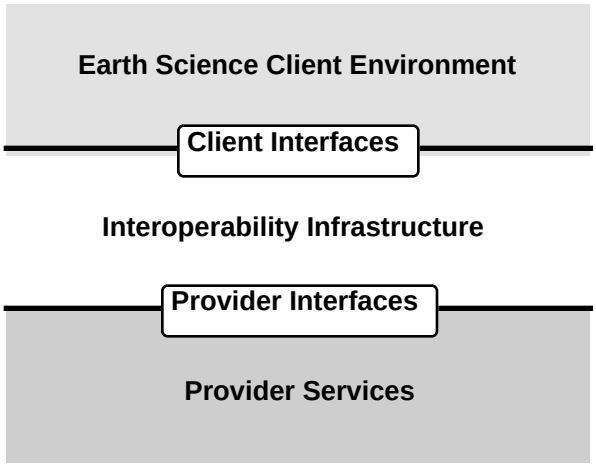

Figure 27. Earth Science Application View

In the OSI 7-layer reference model, the client-server interfaces are layer 7 interfaces. Thus, the SDPS 'client interfaces' and 'provider interfaces' are layer 7 application interfaces, and the user, provider, and interoperation services are (in OSI terminology) 'applications' at the OSI layer 7. They share this layer with the network applications, such as remote procedure call, file transfer, directory services, etc. In essence, and from an SDPS point of view, the OSI layer 7 is divided into two sublayers, as illustrated in Figure 28. For lack of a standard name, we call this the "SDPS Application Layer". The applications in this layer use the network applications layer to communicate with each other (perhaps DCE RPC now and CORBA ORB on top of DCE RPC some time in the future).

---

41 For a description of the client-server reference model, see "ISO 9072-1992: Information Technology - Text Communications - Remote Operations", and "DIS 11578-1 Information Technology - Open Systems Interconnection - Remote Procedure Call - Part 1: Model"

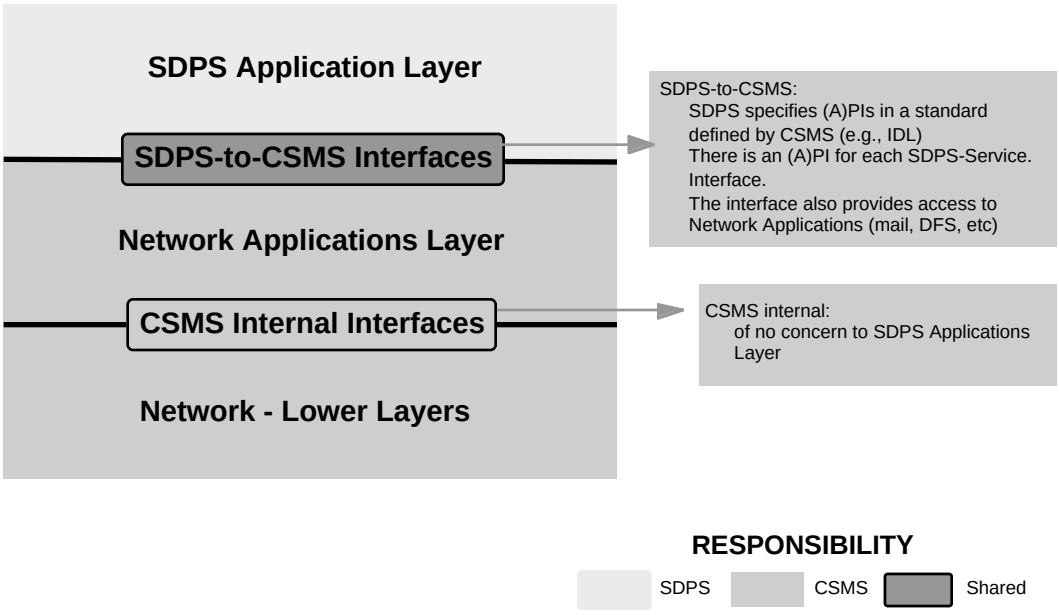

Figure 28. Communications Layering View

6.6.1 Concepts and Reference Model

The network applications fall into the domain of the CSMS segment of ECS. These applications are depicted in the layered protocol reference model shown in Figure 29. CSMS will select the network applications which are appropriate to use, and CSMS will specify the interface which SDPS applications are supposed to use. In today's technology, these interfaces are not merely defined in terms of an API, but also in terms of a language which is to be used to describe and compile the interfaces. In the case of DCE and CORBA, the language is called IDL (note that DCE and CORBA IDL are not identical).

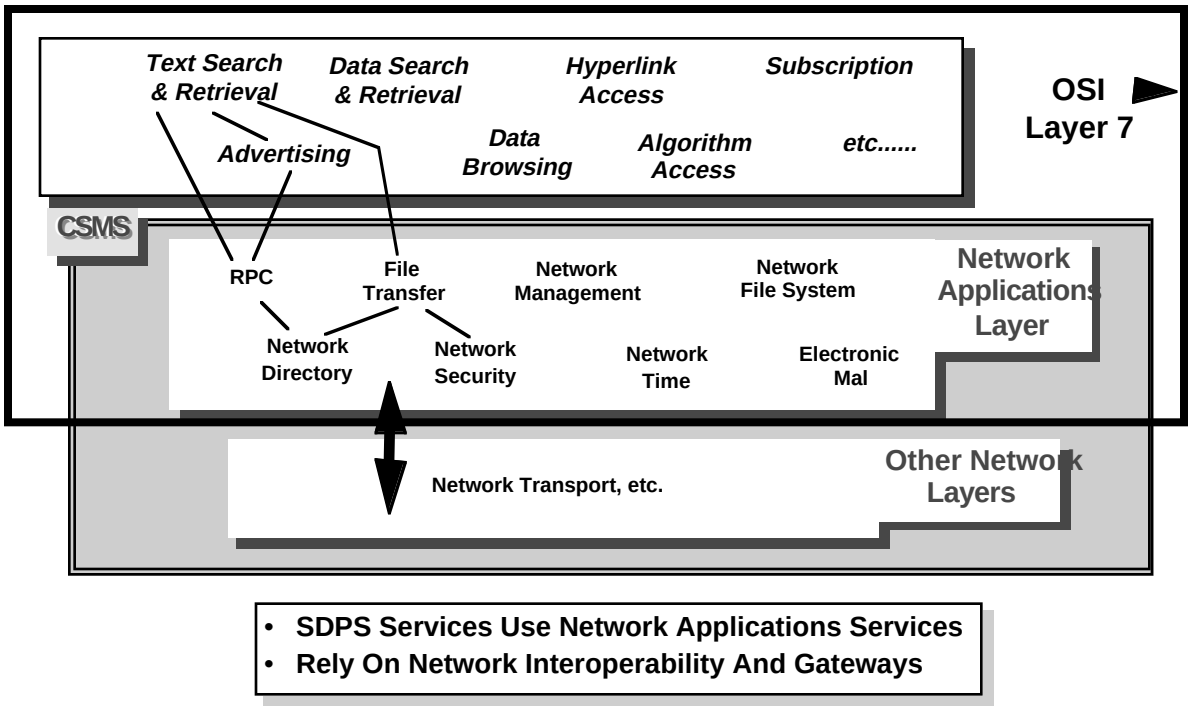

Figure 29. Protocol Reference Model

However, ECS is meant to be an evolvable system. In particular, the communications software infrastructure is likely to be migrated, for example, from DCE in the short term to CORBA in the longer term. CSMS, therefore, will insert an additional layer between the network applications and the SDPS applications. We call the layer a 'wrapping layer' and it is shown in Figure 30. Candidate technologies for such wrapping layers exist today and may be adopted by CSMS, or CSMS may opt to develop its own. In either case, SDPS applications will define interface arguments and behavior, but will do so within the constraints and according to the rules of the wrapping layer provided by CSMS.

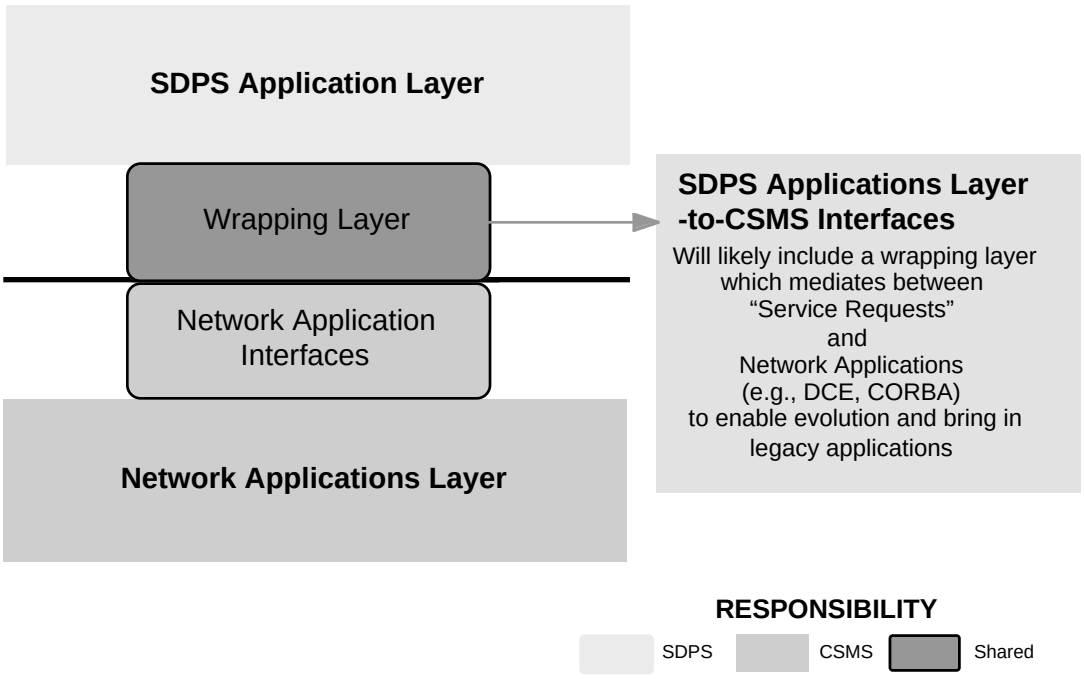

Figure 30. Service Request Wrapper

The various SDPS application layers and their relationship to the communications infrastructure is shown once more in Figure 31. SDPS client applications, such as user interface software will connect via the CSMS wrapping with interoperability and provider services. Implicitly and transparently to them, they will make use of the CSMS network applications to route and possibly translate the requests and their results. Often, the client applications will first use the services of the SDPS Interoperability Applications to find the appropriate SDPS Provider. However, they could communicate directly with a Provider Application if they already know the site and application with which to connect. When Interoperability and Provider Services communicate with and among each other, they also use the network application services.

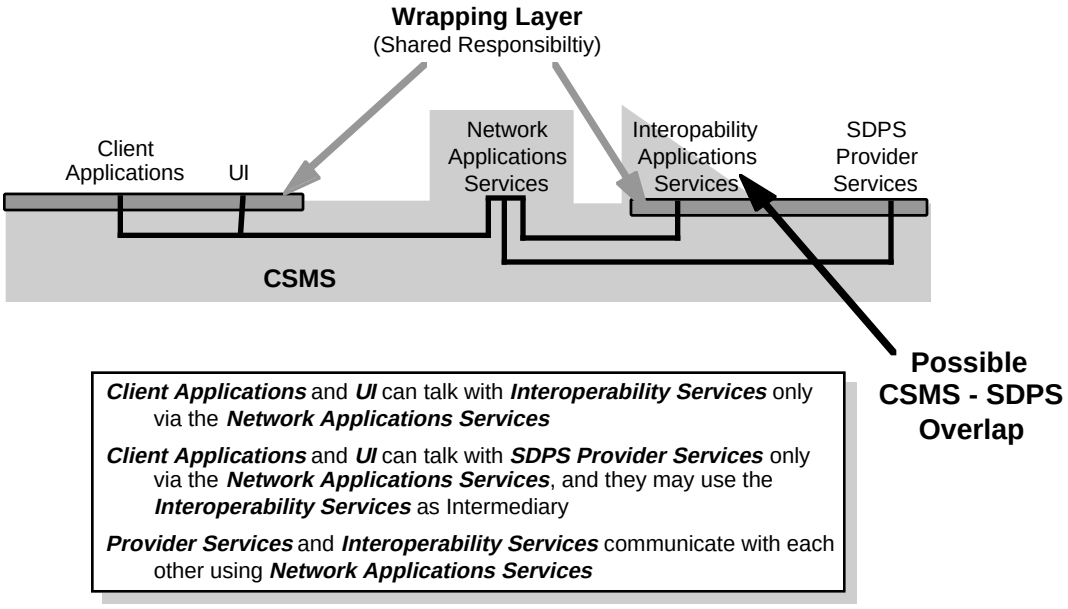

Figure 31. Data Flow View

6.6.2 Implications and Issues

In the following discussion, we explore a number of implications and issues surrounding the Communications and Interconnection Architecture.

SDPS Interoperability Applications could conceivably overlap with CSMS Network Applications. For example, SDPS includes a request broker which shares some characteristics with the Object Request Broker of CORBA[42], and advertising services and request broker services together implement aspects similar to that of a 'trader' in the ODP reference model[43]. This overlap issue will be researched and resolved in the current design phase.

In summary, all three SDPS layers (client, provider, and interoperability layer) operate in layer 7 of the OSI reference model. From a communications perspective, that layer is divided into two sublayers, an SDPS applications layer, and a network applications layer. In the network applications layer CSMS are currently planning to use profile extensions of basic layer 7 services to provide network application services and common facilities for all segments. SDPS applications will communicate with the communications infrastructure via a wrapping interface which will insulate SDPS applications from the specific characteristics and interface requirements of the network applications they use. The wrapping interface and software are as yet to be selected, and the selection will be primarily the responsibility of the ECS CSMS.

---

42 "OMG 91.12.1 Revision 1.1 The Common Object Request Broker: Architecture and Specification."

43 "ISO/JCT1/SC21 N 82.18 Working Draft for Information Technology - Open Distributed Processing - Basic Reference Model - Part I: Overview and Guide to use", September 1992.

6.7 System Management Architecture

The System Management Architecture is based on distributed system management concepts as cooperatively developed by the ISO/CCITT, the Internet Engineering Task Force (IETF), OMNIPoint, OSF DME, and OMG CORBA. The approach to the system management architecture is based on the development of fully distributed, federated services that can be configured as required to strike the proper balance between centralized management and distributed control and authority, in accordance with management policy. Through this policy neutral-architecture, correct engineering practice can be applied to avoid any single point of failure in the system management

Through the system management architecture, components can provide their service to any number of others, not just its "parent" in the hierarchy. Components can establish peer-to-peer relationships without the need for global namespace maintenance. All names are relative to the context in which they are used.

The guiding concept in advanced system management is that managed objects effectively manage themselves by means of their own resource management or process controllers. System Monitoring and Control (SMC) functionality is provided to support monitoring and long-term trend analysis and support. Primary system management services include the classical ISO/CCITT services, including fault, configuration, accounting, and performance services. These services rely on underlying system management function (SMF) services further defined in the ISO/CCITT object models and object management services defined as part of the OMG object models.

6.8 Flight Operations Architecture

The Flight Operations Architecture is currently being developed within the Flight Operations Segment (FOS) of ECS. While the architecture is largely self-contained, there are, nevertheless, some key interactions between FOS and SDPS components. These interactions are primarily organized around presenting users with data acquisition plans/status and allowing data acquisition plans to be modified either by the creation of new acquisition requests or altering existing requests . It is envisaged that these services will be advertised to the system in the same way as all other SDPS services. The clients for these services will be used in conjunction with other services which assist the user in associating data products and/or geophysical parameters with the instrument(s) which need to be scheduled, as well as recommend and allowable instrument parameter settings.

As the overall ECS architecture develops further, the architecture for the interaction between FOS and SDPS will be further defined.

7. Risk

The discussion of the logical architectures in section 6 presented a number of issues associated with various aspects of the emerging architectural design. These include issues of standards, unsure or changing requirements, maturity of COTS software, and others.

In this section, we attempt to summarize some key risk areas and our current approaches to mitigation. The discussion is organized into the three layers of the conceptual architecture: Client, Interoperability, and Provider, with risk and mitigation strategies discussed by topic. This organization provides an application-oriented view that goes across the logical architecture decomposition presented in the previous section. In the future, it may make sense to try and map these risks and mitigation strategies to particular logical architecture decompositions.

7.1 Client Layer

Following are the key risk considerations in the client layer of the architecture:

- integration of DAAC unique specialized interfaces

  The current concept for the user interface is a workbench which can deal with three different types of interface - general, specialized, and object. The key risks will be the development of the APIs which support the workbench to allow specialized interfaces to be linked to the workbench through dynamic linking.

  Mitigation: currently looking for equivalent functionality in other systems to be accompanied by prototyping

- *resolution of incompatibility between service and client*

  The flexibility of the new architecture permits providers (DAACs and SCFs) to offer new services which are different from the common ECS services such as search, browse etc. While this environment brings many advantages it will inevitably lead to some incompatibility between a specific users client and some of the services they wish to access. The client API must be able to assist the user in overcoming the incompatibility.

  Mitigation: currently looking for equivalent functionality in other systems to be accompanied by prototyping

7.2 Interoperability Layer

Following are the key risk considerations in the interoperability layer of the architecture:

- Earth Science Data Language

  A sophisticated system for handling request messages and the results references is required. The definition of a single ECS specific language is not feasible or practical, and there are no approaches in the existing published standards. A flexible approach which

builds on existing standards, and is able to deal with precise and imprecise queries is required.

Mitigation: prototype planned

- identification of tool (client) and service compatibility

  One of the key risks in the client layer is handling the possibility of incompatibility between the client/request and the most relevant service. An important part of the handling this issue, is identifying the potential incompatibility. This will impact the definition of the request message, the service advertising and the data dictionary/vocabulary services.

  Mitigation: working with V0 team on vocabulary/data dictionary services; proposing prototypes for the ESDL and advertising service

- optimization of search queries

  The optimal partition of an intersite request is a complex problem, which is currently in the realm of research. A complete solution to this problem will not be available for the first releases of ECS, though it is clear that there will be some developments in this area during the project time period because it is relevant in many other domains.

  Mitigation: identifying appropriate external research which ECS could contribute to; planning an evolution from simple 'user-assisted' optimization, through increasing atomization of the process

7.3 Provider Layer

Following are the key risk considerations in the provider layer of the architecture:

- cost of processing power

  The significant increase in processing requirement (75 * the baseline) will require considerable rethinking of the processing approach. It may not be possible to achieve this within a reasonable budget without changing some of the constraints that impact the processing (e.g., distribution, archiving, reprocessing capability.

  Mitigation: primary focus of performance modeling in next few months

- relationship between processing power and algorithm development approach

  The increase in processing requirement will also require the use of less standard hardware (e.g., MPP, workstation clusters). The current science algorithm I&T concept involves the scientists developing on SCFs and delivering to ECS for I&T. This approach is reasonable if the SCF and DAAC processors are of the same architecture. There will be considerably more risk if the architectures are different. The only practical response would be to either plan on considerable reworking of the algorithm at the DAAC, or allow the algorithm developers extensive time on the target machine (this will involve more cost). Moreover, it is highly unlikely that the optimum processing capacity will be achieved with this approach to algorithm development

Mitigation: once the processing architecture has been reestablished, revisit the algorithm development approach with each development team

- DBMS technology

  The demands of the ECS services for a DBMS cannot be met by current RDBMS solutions. Other DBMS technologies, while offering some potential, are considerably less mature. Being a high cost solution the scope of selecting the mature technology and replacing it later may not be available to the project.

  Mitigation: investigation of OODBMS and ORDBMS underway - prototyping planned

- Hierarchical Storage Management

  A key part of the provider architecture is the connection from each of the services (accessed through a data server) to the archive - the Hierarchical Storage Management element. With the move towards users "accessing" data rather than simply ordering the data the requirements on this element have become more sophisticated. The original baseline - UniTree - has been found to be deficient in several areas

  Mitigation: prototyping and planning to work with vendors of most promising products to develop appropriate COTS. Design will be as independent of the HSM as possible so that technology evolution can be accommodated

- process scheduling

  The scheduler was originally only involved in science processing/reprocessing. The new architecture now requires scheduling of many access service processing. The optimum solution would be to identify a single approach to all scheduling requirements.

  Mitigation: currently identifying all implied requirements for the scheduling, to be followed by prototyping

- system wide management coordination

  The new architecture allows more autonomy for DAACs in terms of the services they provide and how they organize their data collections. This makes the issue of coordinated system management between DAACs more complex.

  Mitigation: CSMS prototype

8. Future Directions

The sub-architectures described in section 6 provide a multi-faceted representation of ECS. In order to facilitate system decomposition and design, we will continue to develop a layered reference model that provides different levels of abstraction of the system. This reference model supports continual refinement of the system decomposition put forth in this paper, leading up to the System Design Review in mid-1994, and beyond as the design evolves. Figure 32 shows the various layers of the current reference model.

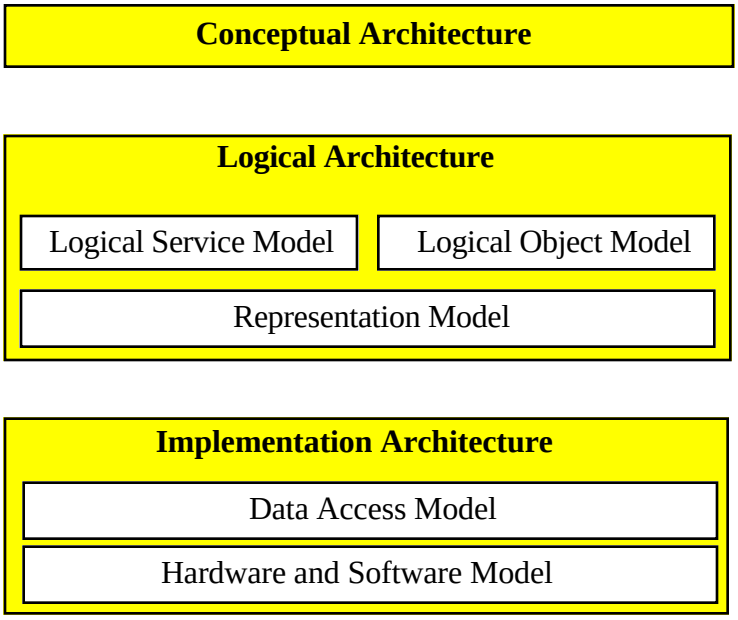

Figure 32. Layered Reference Model

The various components of the reference model are defined below:

Conceptual Architecture

The conceptual architecture represents a high level model of the system, based on the drivers and key architectural principles. The conceptual architecture includes organizational and functional descriptions of the system using the 3-layered user-provider model.

Logical Architecture

The logical architecture includes a decomposition of the system into an objects- and services-based representation. The logical architecture uses the sub-architecture framework defined here, developed to a greater depth and referencing services and objects shared across sub-architecture boundaries. The logical architecture includes the following reference models:

- Logical Service Model

  The logical service model focuses on the system services available to "client" applications. In this case, we mean both data providers (who are clients of fundamental system services), and users. Users are taken here to imply both human and computer-based (algorithmic analysis) clients of system services. This is the level addressed in this document.

- Logical Object Model

  The logical object model provides for the definition of additional internal system services, and the organization of both internal and client services around a set of system objects that will provide these services.

- Representation Model

  The representation model presents a mapping of the logical object model to a software representation – functions and interfaces, for a functional decomposition, or objects and services, for an object-based or object-oriented decomposition.

Implementation Architecture

The implementation architecture maps the logical components developed in the logical architecture, into physical implementations of both data and processing. The implementation architecture includes the Hardware Modelwhich presents a physical assignment of components at the higher levels onto hardware components distributed throughout the system.

At SDR, some work will have been completed in all layers of this model so that the design can be adequately reviewed. Following the SDR work will continue to refine and extend the detail of the design, particularly in the Representation and Data Access Models, leading towards the PDR for Release A.

9. References

1 Pochardt-Johnson, C., “A Guide to ODP”, ISO/IEC JTC1.21.43, Document X3T3/SD-4, August, 1992.

2 ISO 9072-1992: Information Technology - Text Communications - Remote Operations

3 Reference Model of Data Management (RM-DM), ISO 10032:1994 (SC21/WG3 N1438.)

4 DIS 11578-1 Information Technology - Open Systems Interconnection - Remote Procedure Call - Part 1: Model

5 OMG 91.12.1 Revision 1.1 The Common Object Request Broker: Architecture and Specification.

6 “ISO/JCT1/SC21 N 82.18 Working Draft for Information Technology - Open Distributed Processing - Basic Reference Model - Part I: Overview and Guide to use”, September 1992

7 IEEE Storage System Standards Working Group (SSSWG, Project 1244), Reference Model for Open Storage Systems Interconnection, Mass Storage Systems Reference Model Version 5, Unapproved Draft 1.2, October 18, 1993.

8 “EOSDIS Core System (ECS) Requirements Specification”, ECS Document # 193-216-SE1-001, August, 1993 (in revision).

9 “Standards and Procedures for the ECS Project”, ECS Document # 193-202-SE1-001, August, 1993.

10 “ECS Operations Concept Document for the ECS Project”, ECS Document # 193-604-OP1-001, August, 1993 (in revision).

11 “Science-based System Architecture Drivers for the ECS Project” white paper, ECS Document # 193-00611, December, 1993.

12 “ECS Evolutionary Development White Paper”, ECS Document # 193-00623, December, 1993.

13 “GCDIS / UserDIS Study” white paper, ECS Document # 193-000626, January, 1994

14 “ECS Science Requirements Summary” white paper, ECS Document # FB9402V1, February, 1994.

15 “Version 0 Analysis Report”, ECS Document # 194-206-SE2-001, Working Draft, February, 1994.

Abbreviations and Acronyms

ADC	Affiliated Data Center
AI	artifical intelligence
ANSI	American National Standards Institute
API	application programmer's interface
ASCII	American Standard Code for Information Interchange
ASTER	Advanced Space borne Thermal Emission and Reflection Radiometer
AVHRR	Advanced Very High Resolution Radiometer
CEOS	Committee on Earth Observation Satellites
CINTEX	CEOS Inventory Interoperability Experiment
CORBA	Common Object Request Broking Architecture
COTS	commercial off-the-shelf (hardware or software)
CSMS	Communications and System Management Segment
DAAC	Distributed Active Archive Center
DBMS	data base management system
DCE	Distributed Communications Environment of OSF
E-mail	electronic mail
ECS	EOSDIS core system
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
ESA	European Space Agency
ESDIS	Earth Science Data and Information System
ESIS	European Space Information System
FIFE	First ISCLSCP Field Experiment
FITS	Flexible Image Transfer System
FOS	Flight Operations Segment
GC	Global Change
GCDIS	Global Change Data Information System
GCRP	Global Change Research Program

GIF	Graphical Interchange Format
GSFC	Goddard Space Flight Center
GUI	Graphic User Interface
H/W	hardware
HITC	Hughes Information Technology Corporation
HDF	hierarchical data format
HMI	human machine interface
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronic Engineers
IMS	Information Management System (ECS)
IP	International Participant/Partner
ISO	International Standards Organization
JGOFS	Joint Global Ocean Forecasting System
L0-L4	Level 0 through level 4 (processing)
LaRC	Langley Research Center
kbytes	Kilo-bytes (10^3)
Mbytes	Mega-bytes (10^6)
MoU	memorandum of understanding
MTPE	Mission to Planet Earth
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputer Applications
netCDF	network version of the Common Data Format
NIIT	National Information Infrastructure (NII) Testbed
NRC	National Research Council
NRT	near real time (data)
OODBMS	object oriented database management system
ODC	other data center
ODL	Object Description Language
OS	operating system
OSF	Open Systems Foundation
OSI	Open System Interconnect

PI	principal investigator
POSIX	Portable Operating System Interface for Computer Environments
QA	quality assurance or quality assessment
RDBMS	relational database management system
RFQ	request for quote
RPC	remote procedure call
S2000	Sequoia 2000 project
S/W	software
SCF	Science Computing Facility
SDPS	Science Data Processing Segment
SFDU	Standard Formatted Data Unit
SMC	System Management Center (ECS)
SQL	Structured Query Language
UIT	User Interface Terminal (ESA)
UNIDATA	University Data System (NSF - Atmospheric Sciences Division)
UserDIS	Data Information System based on providers in the earth science user community
V0	Version Zero (EOSDIS)
WAIS	Wide Area Information System
WOCE	World Ocean Circulation Experiment
WWW	World Wide Web
X.500	OSI standard for directory services
XBT	Instrument for measuring temperature and salinity profiles with depth in the ocean
X^nDIS	an architectural concept to satisy EOSDIS, GCDIS and UserDIS needs